目　录

第1章　MATLAB仿真技术与应用简介

MATLAB经过十几年的发展完善，已经成为国际控制界公认的标准计算软件，并在大学里广泛使用，深受大学生的喜爱。在欧美的一些大学里，MATLAB已经成为理工科大学生、硕士生、博士生必须掌握的基本工具之一。MATLAB在仿真领域已经成为主流工具，而且MATLAB能够与各种程序语言进行混合编程，大大加快了实际开发周期，这也是它广泛应用于仿真领域的又一原因。

本章主要内容：

- 系统仿真技术概述
- MATLAB仿真技术的发展与应用
- MATLAB仿真技术的特点
- 仿真应用实例简介

1.1　系统仿真技术概述

自从有人类以来，人们为了满足自身的基本需要，一直在同外部环境发生着联系。随着时间的流逝，人类所依赖的这种联系方式变得日趋复杂并多样化。人类在科学和工程技术上所做的研究就是努力理解真实世界并能掌握与真实世界发生联系的形式。随着科学和工程技术的发展，人们认识自然和改造自然的能力和手段也不断增强。回顾科学和工程技术的发展历史，在计算机出现之前，科学研究中的绝大部分工作是利用数学手段或其他方法对事物或真实世界进行描述，这也就是建模活动。计算机的出现对科学和工程技术的发展产生了深远的影响，使人们能对复杂事物和复杂系统建立模型并利用计算机进行求解，这些手段和方法逐步形成了计算机仿真技术。建模与仿真成为当今现代科学技术研究的主要内容，建模与仿真技术也渗透到各学科和工程技术领域。

建模与仿真这一领域的发展可分为两个阶段。其一是计算机出现之前，主要是在物理科学基础上的建模；其二是20世纪40年代计算机诞生以后，出现了计算机仿真技术，它的发展也促进了建模技术的发展，建模与仿真日益紧密，互不可分。

计算机问世不久，人们就清楚地看到这种新机器给很多问题的求解带来了异常的生机，但最初计算机在人类的科学工程活动中的作用是有限的，它仅仅作为一部强有力的、高速的、不会说话的机器。随着计算机的日益完善，许多复杂的模型可以通过计算机来进行计算求解，它在科学技术中的作用也与日俱增，并把模型求解的手段发展成为现代的计算机仿真技术，这样才真正诞生了“仿真”这个新词。计算机仿真技术有着巨大的优越性，利用它可以求解许多复杂而无法用数学手段解析求解的问题，利用它可以预演或再现系统的运动规律或运动过程，利用它可以对无法直接进行实验的系统进行仿真试验研究，从而可以节省大量的资源和费用。由于计算机仿真技术的优越性，它的应用领域已经非常广泛，而且越来越受到普遍的重视。

“仿真”一词即模仿真实，多数人认为仿真就是程序的运行，该程序表示了一个抽象的模型，用来研究现实系统的一些特征。在这个意义上，仿真活动可认为就是支持模型建立与模型分析的所有计算，它对科学工程方法中的模型建立阶段和模型分析阶段具有同等重要的价值。

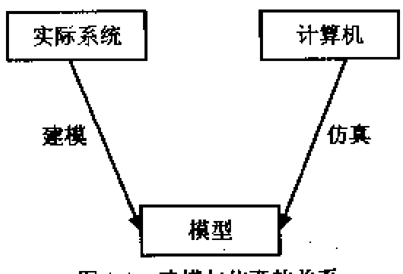


图 1-1　建模与仿真的关系

建模与仿真是指构造现实世界实际系统的模型和在计算机上进行仿真的有关复杂活动，它主要包括实际系统、模型和计算机等 3 个基本部分，同时考虑 3 个基本部分之间的关系，即建模和仿真的关系，如图 1-1 所示。

建模关系主要研究实际系统与模型之间的关系，它通过对实际系统的观测和检测，在忽略次要因素及不可检测变量的基础上，用数学的方法进行描述从而获得实际系统的简化近似模型。仿真关系主要研究计算机的程序实现与模型之间的关系，其程序能为计算机所接受并在计算机上运行。

仿真关系主要关注的是计算机执行模型所规定的指令的真实性。一个模型的程序能否真实地体现模型所具有的内涵，称之为程序的正确性。它主要关心由计算机产生数据的准确性，要确认计算机是真实地产生实际模型的行为还是产生错误的输出，确认计算机是执行了模型本身的特征还是只是一个假象。要验证模型的有效性，需要把模型的行为同实际系统的行为进行比较。这样，首先要检验模型程序的正确性，才不会把程序的问题和模型的问题混淆起来。这也要求我们必须懂得仿真过程，包括仿真机理和仿真策略。除了关注程序的正确性外，仿真关系也包括模型的仿真机理和仿真策略研究。

1984 年，Oren 提出仿真是一种基于模型的活动，并认为仿真包括了 3 个基本要素：对仿真问题的描述、行为产生器和模型行为及其处理。

1．对仿真问题的描述

任何一个仿真问题都由模型与实验两部分组成，这一点与传统的仿真定义是完全一致的。而任何一个数学模型，不论采用什么样的建模方法，又都由两部分组成：一个参数模型及一组参数值。当我们给定了一个参数模型，同时又赋予它具体的参数值后，就形成了一个特定的模型。另外，实验也可分为两部分：实验框架及仿真运行控制。

一个实验框架可以定义为一组条件，在该条件下，系统可被观测或被进行实验。具体来讲，试验框架可由 5 个部分组成：可观测变量，输入值的调度，初始设置，终止条件，对数据的采集及压缩的具体说明。

所谓对数据的压缩，是指将某个描述变量的轨迹行为转变为更浓缩的形式。一般可分为统计数据压缩及解析数据压缩两种，前者是讲轨迹行为转变为几个点行为，如最小值、最大值、取值范围、标准差等，后者则是讲轨迹行为转变为曲线或某种解析形式。

2．行为产生器

行为产生器是一套对模型进行实验的软件，比如连续系统仿真中的仿真计算程序。由它可以产生一组随时间变化的系统状态变量的数据（称为模型行为）。

3．模型行为及其处理

模型行为有 3 种类型：点行为、轨迹行为及结构行为。

在各种类型的仿真中可以获得轨迹行为，它通常被表示为一组系统中各种描述变量随时间推移而变化的数据。

结构行为只可以从可变结构系统模型中获得，即这种系统模型的静态结构（指定一描述变量及其属性）及动态结构（指状态变量之间的动态关系）是可变的。

点行为则是指模型行为的一种特定属性，如最小值、最大值、振荡次数、上升时间、稳定时间等。一般来讲，常规的仿真软件并不产生点行为，它是在对数据进行压缩后才能产生出来。

行为处理包括对行为进行分析及显示。

根据以上分析可以清楚地看出，整个仿真过程包含了建模、实验和分析 3 个主要步骤，它们分别对应上述 3 个要素。这种规范化的认识形成了仿真的基本概念框架——“建模—实验—分析”三段式。

一般来讲，过去的仿真一直着重于“试验”，也即着重于如何获得系统中有关变量的时间响应。近 20 年来，对模型的研究及如何来建立模型已成为仿真活动所关心的问题。同时，建模与模型实验这两者本来是不可分割的，应该一起进行研究。因此，现代仿真是这样一个过程，它为了分析与研究存在的或尚未建成的系统，先建立该系统的模型，再将其安放到计算机上进行实验。仿真的重点包括“模型”和“实验”两个方面，仿真是一种基于模型的活动，仿真的基本概念框架为“建模—实验—分析”。

现代仿真技术的发展是与控制工程、系统工程和计算机技术的发展密切相关的。控制工程是仿真技术较早应用的领域之一，控制工程技术的发展为现代仿真技术的形成和发展奠定了良好的基础。系统工程的发展进一步完善了系统建模与仿真的理论体系，同时使系统仿真广泛应用于非工程系统的研究和预测。

计算机仿真技术不论是在理论上还是在实践上都已经取得了丰硕的成果，积累了大量的系统仿真模型和行之有效的仿真算法。但仿真技术目前仍然存在一些缺陷，例如建模方法尚不完善，研究同一个系统的同一个问题一样建立出不同的模型，而且有效社会经济系统中的问题尚无法建立准确的模型来进行求解。同时，检测者必须通过建模者和仿真试验人员才能介入到对系统的仿真分析中。随着建模与仿真的理论和方法的研究不断深入，以及作为其支撑技术之一的计算机技术的不断发展和进步，计算机仿真技术在应用过程中出现的问题将逐步得到解决。进入 21 世纪，计算机技术的各个方面都取得了异乎寻常的进展。微处理器性能的增长时段利用微型计算机和工作站进行复杂系统的仿真分析成为可能，当然像中长期天气预报这样模型复杂、数据繁多、实时性要求高的问题的计算仍然离不开巨型机。在软件设计中广泛采用了面向对象的思想和方法，再加上计算机图形技术的进步，仿真过程中的人机交互越来越方便、直观。总之，计算机仿真技术正朝着一体化建模与仿真环境的方向稳步发展。下面列出在仿真领域的 6 个研究方向，作为本节的结束。

- 建模方法学（Modeling Methodology）
- 面向对象仿真（Object-Oriented Simulation）
- 分布交互仿真（Distributed Interactive Simulation）
- 人工智能（Artificial Intelligence）与计算机仿真
- 虚拟现实（Virtual Reality）仿真
- Internet 网上仿真

1.2 MATLAB 仿真技术的发展与应用

MATLAB 诞生在 20 世纪 70 年代，它的编写者是 Cleve Moler 博士和他的同事。当时，Cleve Moler 博士和他的同事开发了 EISPACK 和 LINPACK 的 Fortran 子程序库，这两个程序库主要是求解线性方程的程序库。但是，Cleve Moler 发现学生使用这两个程序库时有困难，主要是接口程序不好写，很费时间。于是 Cleve Moler 自己动手，在业余时间里编写了 EISPACK 和 LINPACK 的接口程序。Cleve Moler 给这个接口程序取名为 MATLAB，意为矩阵（MATRIX）和实验（LABORATORY）的组合。

1984 年，Cleve Moler 和 John Little 成立了 Math Works 公司，正式把 MATLAB 推向市场并继续进行 MATLAB 的开发。1993 年，Math Works 公司推出了 MATLAB4.0；1995 年，Math Works 公司推出 MATLAB4.2C（For Windows3.x）；1997 年推出 MATLAB5.0；2000 年 10 月，Math Works 公司推出 MATLAB6.0；2002 年 8 月推出 MATLAB6.5；2004 年 6 月最新的版本 MATLAB7.0 开始发布。每一次新版本的推出都使 MATLAB 有了长足的进步，界面越来越友好，内容越来越丰富，功能越来越强大。

在 MATLAB 以商品形式出现后，仅短短几年就以良好的开放性和运行的可靠性淘汰了当时众多的软件包。进入 20 世纪 90 年代后，MATLAB 已经成为国际控制界公认的标准计算软件，并在大学里广泛使用，深受大学生的喜爱。在欧美的一些大学里，诸如应用代数、数理统计、自动控制、数字信号处理、模拟与数字通信、时间序列分析、动态系统仿真等课程的教科书都把 MATLAB 作为内容，在那里 MATLAB 成为攻读学位的大学生、硕士生、博士生必须掌握的基本工具之一。

MATLAB 长于数值计算，能处理大量的数据，而且效率比较高。Math Works 公司在此基础上开拓了符号计算、文字处理、可视化建模和实时控制能力，增强了 MATLAB 的市场竞争力，使 MATLAB 成为了市场主流的数值计算软件。经过多年的工程实践，人们已经发现 MATLAB 作为计算工具和科技资源可以扩大科学研究的服务，提高工程生产的效率，缩短开发周期，加快探索步伐，激发创作活力。

Simulink 作为 MATLAB 工具包中的重要一员，是一种图形化的仿真工具包，能够进行动态系统建模、仿真和综合分析，可以处理线性和非线性系统，离散、连续和混合系统，以及单任务和多任务系统，并在同一系统中支持不同的变化速率。

Simulink 应用领域非常广，可使用的领域包括航空航天、电子、力学、数学、通信、影视、核控制等。世界各地的工程师都在利用它来对实际问题建模，解决问题。

1.3 MATLAB 仿真技术的特点

应用 MATLAB，尤其是应用其工具包 Simulink 进行仿真已经成为当今仿真领域的主流方法，它的应用如此之广泛，是因为它具有与众不同的特点，下面介绍主要特点。

1. 交互式仿真工具

Simulink 具有非常友好的界面，提倡将模型通过框图的形式表示出来，允许随意修改模

块的参数，允许用户将已有的模型添加组合到一起，或者将自己创建的模块添加到模型当中，并且可以直接无缝地使用 MATLAB 的所有分析工具，对最后得到的结果进行分析，并能够将结果可视化显示。Simulink 的一个意图就是让用户在使用 Simulink 的同时能够感受到建模与仿真的乐趣。通过这个平台，可以激发用户不断地提出问题，对问题进行建模。

对建模，Simulink 提供了非常方便的图形建模方式，通过单击和拖放鼠标搭建框图来完成仿真模型的建立，实现所见即所得。通过 Simulink 提供的窗口，搭建框图就如同用铅笔在白纸上画图一样方便、快捷。与以前将微分方程写成某种语言或程序相比，这无疑是一个创举。

在建立模型之后和允许仿真之前，必须对模型进行参数设置。仿真所需要的模型参数可以通过 Simulink 菜单来进行设置，方便快捷。

2. 良好的移植性和扩展性

MATLAB 是用 C 语言写的，由于 C 语言的良好移植性，因而 MATLAB 也可以方便地移植到能运行 C 语言的平台上。适合 MATLAB 的工作平台有：Windows 系列、UNIX、Linux、VMS6.1、PowerMac。除了内部函数外，MATLAB 所有的核心文件和工具箱的文件都是公开的，都是可读可写的源文件，用户可以通过对源文件进行修改或自己编程构成新的工具箱。

3. 易学易用

MATLAB 是一种面向科学计算与工程计算的高级语言，允许用数学形式的语言编写程序，被称为第四代计算机语言，比 C/C++、Fortran、Basic 等语言更加接近人们书写计算公式的思维方式，用 MATLAB 编写程序犹如在演算纸上排列公式和求解问题。因此，MATLAB 语言也可通俗地称为演算纸式科学算法语言。由于它编写简单，所以编程效率高，易学易用。

4. 方便的可视化

MATLAB 的绘图是十分方便的，它有一系列绘图函数，例如象限坐标、对数坐标、半对数坐标、极坐标等，均只需要调用不同的绘图函数，在图上标出图题、*x*/*y* 轴标注，格绘制也只需要调用相应的命令，简单易行。另外，在调用绘图函数时调整自变量可绘出不同颜色的点、线、复线或多重线，这种为科学研究着想的设计，通过其他编程语言实现都是比较麻烦的。

1.4 仿真应用实例简介

仿真在现实世界和科学研究中有广泛的应用，按其应用的实际系统类型可分为：离散时间系统、连续时间系统和离散事件系统等。本节列举一个简单的离散时间系统实例——污染模型仿真，以给读者一个仿真的初步认识，在以下的章节中将逐步深入到如何应用 MATLAB 进行系统仿真。

Forester 在 1972 年提出了一个世界模型，这个模型企图解释关于污染和自然资源耗尽对全球人口数量和质量的影响。图 1-2 的污染模型是世界模型的一部分，图中每个圆圈和方框均表示某个函数，圆圈表示瞬时函数，即根据时间 t 的各输入值确定模型在时间 t 的输出值

的函数。例如，函数 *POLCMT* 描述了人均产生的污染与人均资本投资增值率的关系，它体现了资本投资的量越大，产生污染也就越大。若在时间 t，*CIR* 有 x 值和 *POLCM* 有 y 值，则有 $y=POLCMT(X)$,也可简化为 $POLCM(t)=POLCMT(CIR(t))$。方框表示记忆或非瞬时函数，根据模型被描述为连续时间模型还是离散时间模型，这些方框可以表示积分器或离散时间模拟装置。下面从离散时间模型角度出发来推导污染模型的形式描述。

设 $POL(t+h)=POL(t)+[POLG(t)-POLA(t)]h$

这就是说，在时间 $t+h$ 的 *POL*（污染）（h 是离散时间步长，例如 1 年）等于时间 t 的 *POL* 加上 $POLG(t)h$（在时间 t 产生的污染）减去 $POLA(t)h$（在时间 t 被吸收的污染）。换句话说，$[POLG(t)-POLA(t)]h$ 是$(t,t+h)$期间产生的净污染量，加上这个期间开始时的污染量 $POL(t)$，便得到该期间末污染量 $POL(t+h)$。$POLG(t)-POLA(t)$是产生污染的净比率（近似于连续时间模型中污染 *POL* 的导数率）。

由图 1-2 可对求解污染 $POL(t+h)$的方程进一步展开，并可表示成时间 t 的状态变量值的函数。整个过程的展开方程式包括：

$$POL(t+h)=POL(t)+[POLG(t)-POLA(t)]H$$

$$POLG(t)=P(t)\times POLCM(t)$$

$$POLCM(t)=POLCMT(CIR(t))$$

$$CIR(t)=\frac{CI(t)}{P(t)}$$

$$POLA(t)=POL(t)\times POLR(t)$$

$$POLR(t)=POLCRT(POL(t))$$

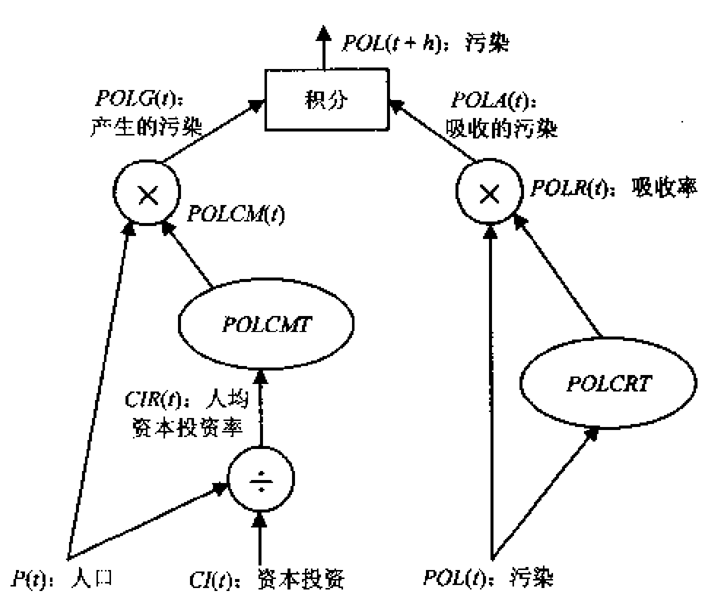


图 1-2　污染模型图

这样，时间 $t+h$ 的 *POL*（污染）可表示为状态变量 *POL*（污染）、*CI*（资本投资率）和 *P*（人口）在时间 t 值的函数，即：

$$POL(t+h)=POL(t)+[P(t)\times POLCMT(\frac{CI(t)}{P(t)})-POL(t)\times POLCRT(POL(t))]h$$

状态变量 *POL* 的转移函数可给定为：

$$\delta_{POL}(pol,ci,p) = pol + [p \times POLCMT(\frac{ci}{p}) - pol \times POLCRT(pol)]h$$

如果在时间 t，POL、CI 和 P 有 pol、ci 和 p 值，那么在时间 $t+h$，POL 有 $\delta_{POL}(pol,ci,p)$ 值。

综上所述，对于已经标示了瞬时函数及记忆函数的模型网络描述，每个模型网络图可容易地转换成某些仿真语言的语句，或某些图形化仿真工具的对应模块，如 Simulink。另一方面，由模型网络描述也可容易地明确状态转移函数和输出函数。

以上向读者简单介绍了一个仿真应用实例，以期给大家一个大概印象，在下面的章节中，我们将逐步学习如何运用 MATLAB 的 Simulink 进行仿真。

第 2 章　Simulink6.0 快速入门

Simulink 是一种图形化的仿真工具包，能够进行动态系统建模、仿真和综合分析，可以处理线性和非线性系统，离散、连续和混合系统，以及单任务和多任务系统，并在同一系统中支持不同的变化速率。本章主要带领读者快速熟悉 Simulink，初步掌握 Simulink 的使用方法。

本章主要内容：

- Simulink 简介
- Simulink6.0 快速入门
- Simulink 是如何工作的

2.1　Simulink 简介

Simulink 是 MATLAB 众多工具包中的一员，它现在已经成为仿真领域的主流工具，下面将带领读者逐步认识 Simulink。

2.1.1　什么是 Simulink

Simulink 是一个用来对动态系统进行建模、仿真和分析的软件包，它支持线性和非线性系统，连续和离散时间模型，或者是两者的混合。系统还可以是多采样率的，比如系统的不同部分拥有不同的采样率。对于建模，Simulink 提供了一个图形化的用户界面（GUI），可以用鼠标点击和拖拉模块的图标建模。通过图形界面，可以像用铅笔在纸上画图一样画模型图，这是以前需要用编程语言明确地用公式表达微分方程的仿真软件包所远远不能相比的。Simulink 包括一个复杂的由接收器、信号源、线性和非线性组件以及连接件组成的模块库，当然也可以定制或者创建用户自己的模块。所有模型都是分级的，因此可以通过自上而下或者自下而上的方法建立模型。可以在最高层面上查看一个系统，然后通过双击系统中的各个模块进入到系统的低一级层面，以查看到模型的更多细节。这一方法提供了一个了解模型是如何组成以及它的各个部分是如何相互联系的方法。定义完一个模型以后，就可通过 Simulink 的菜单或者在 MATLAB 的命令窗口输入命令对它进行仿真。菜单对于交互式工作非常方便，而命令行方式对于处理成批的仿真比较有用（例如，你在进行 Mont carol 仿真时想使参数遍历某一范围的值）。使用 Scopes 或者其他的显示模块，可以在运行仿真时观察到仿真的结果。另外，还可以在仿真时改变参数，并且立即就可看到有什么变化。仿真的结果可以放在 MATLAB 的工作空间（workspace）中以待进一步的处理或者可视化。

模型分析可使用的工具包括可直接通过命令行方式调用的线性化和整理（trimming）工具，MATLAB 的其他各种工具，以及所有应用程序工具箱。因为 MATLAB 和 Simulink 是集成在一起的，所以用户可以在任何环境的任意点对用户的模型进行仿真、分析或修改。

作为一种图形化的仿真工具包，Simulink 能够进行动态系统建模、仿真和综合分析，可以处理线性和非线性系统，离散、连续和混合系统，以及单任务和多任务系统，并在同一系统中支持不同的变化速率。

Simulink 具有非常高的开放性，提倡将模型通过框图形式表示出来，或者将已有的模型添加组合到一起，或者将自己创建的模块添加到模型当中。Simulink 具有较高的交互性，允许随意修改模块参数，并且可以直接无缝地使用 MATLAB 的所有分析工具。对最后得到的结果可进行分析，并能够将结果可视化显示。Simulink 的一个意图就是让用户在使用 Simulink 的同时能够感受到建模与仿真的乐趣。

Simulink 非常实用，应用领域很广，可使用的领域包括航空航天、电子、力学、数学、通信、影视、核控制等。世界各地的工程师都在利用它来对实际问题建模，解决问题。

2.1.2 Simulink6.0 的新特点

Simulink6.0 新增加了许多新功能，现介绍如下。

模型浏览器：它是一个新的工具，利用它可以迅速地设置、查看、创建、查询、更改 Simulink 模型的所有数据和属性。

模块属性配置集：作为模块属性集的一个别名，每个新模块在创建时以一个默认的配置属性集初始化，用户也可以创建自己的属性配置集。

模型参考：允许一个模块包含其他模块，作为该模块的一个组成部分。

模型工作区：Simulink 为每个模块提供它自己的工作区来存储数据，模块可以访问自己的工作区内数据，也可以访问它们参考模块的数据，以及基模块的数据，比如 MATLAB 的工作区。模型工作区允许为一个模块创建数据而不必担心与其他模块数据发生冲突。

新增 ode14x 算法：这是一个隐式调用的算法，它在某些情况下对一些刚性系统比显式的算法要快。

信号、示波器管理器：该新功能可以从全局管理信号发生器及示波器。

新增 4 种对象类型：Simulink.AliasType、Simulink.NumericType、Simulink.StructType、Simulink.Bus。

在模块、信号、信号传输、执行上下文上都作了改进。

算法循环作了改进。

新增加了一个 level2-API 接口，用于创建基于 M-文件的 Simulink 模块与上一版本 level1-API 比较，该版本支持大部分标准的 Simulink 模块属性，包括支持矩阵信号和非双精度数据类型。

新增漫游察看模块功能，对于超过编辑器可见范围的模块编辑非常实用。具体操作也非常简单，只需在漫游时按下 P 或 Q 键，根据左右手习惯即可，这也充分体现了 MATLAB 人机界面之友好。

2.1.3 Simulink6.0 的安装

在安装 MATLAB 时，选择安装 Simulink 组件，则 Simulink 就随 MATLAB 一起安装了。如图 2-1 所示。

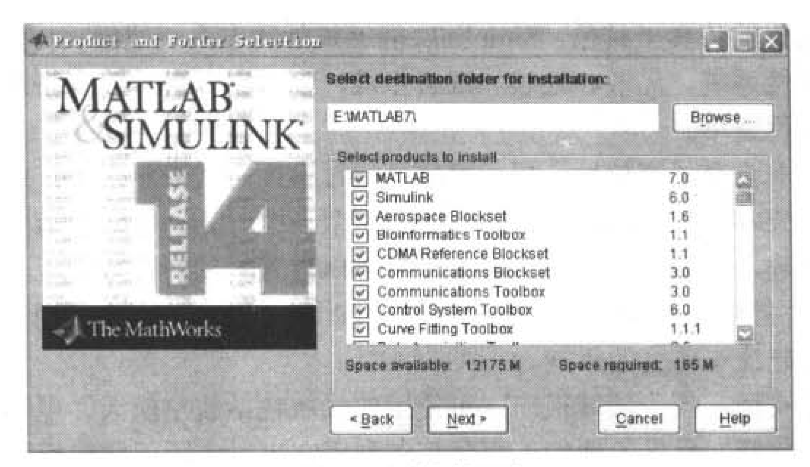


图 2-1 组件安装界面

2.1.4 Simulink6.0 实时工作环境的作用及其主要特点

Simulink 实时工作环境（Real-Time Workshop）直接自动地从 Simulink 模块图生成 C 语言代码，这将允许连续、离散时间或者混合系统的模型可以运行于各种计算机平台，其中包括实时硬件，但 Simulink 是必不可少。

1. 实时工作环境的作用

（1）快速建模

作为一个快速建模工具，实时工作环境使得用户可以快速实现自己的设计，而不用手工编写长长的代码然后进行调试。控制、信号处理和动态系统的算法可以通过开发图形化的 Simulink 模块图，并且自动生成 C 语言源码来实现。

（2）嵌入式实时控制

一旦一个系统已经用 Simulink 设计出来，就可生成实时控制器或数字信号处理器的代码，然后可对代码进行编译、链接，最后装载到目标处理器中。实时工作环境支持 D5P 板、嵌入式控制器，以及多种用户和商业开发的硬件。

（3）实时仿真

对循环中硬件仿真，可以为整个系统或指定的分系统创建和执行代码。典型的应用包括训练仿真器、实时模型验证和测试。

（4）单机仿真

单机仿真可以在你的主机上直接运行或者传送到另外的系统上以远程方式执行。由于时间历史数据被以二进制或 ASCII 文件保存在 MATLAB 中，可以很容易地被装入 MATLAB 中，以待进一步进行分析或图形显示。

2. 实时工作环境的主要特点

实时工作环境具有一系列复杂的能力和特性，以提供实现各种应用的灵活性。

（1）自动代码生成，以处理连续时间、离散时间和混合系统。

（2）优化的代码，以保证快速执行。

（3）控制框架结构应用程序接口（API）自动地使用定制的 make 文件来创建和下载

Object 文件到目标硬件上。

（4）可移植的代码使其应用环境更加广泛。

（5）简明、可读并具有详细注释的代码使得维护非常简单。

（6）从 Simulink 下载到外部硬件上的交互参数使系统在工作状态下很容易调整。

（7）一个菜单驱动的图形用户界面使得软件的使用非常容易。

2.1.5 Simulink6.0 工作环境

Simulink 工作环境经过几年的发展，已经成为在学术和工业界用来建模和仿真的主流工具包。

Simulink 会激发你尝试自己的各种想法，你可以非常容易地根据自己的想法建立模型，或在其中添加一个现有的模型。Simulink 的仿真是交互形式的，所以你可以在仿真期间改变其参数，而立即看到改变的影响，在 Simulink 中可以和 MATLAB 的其他分析工具无缝结合，结果可以被立即可视化，使你在建模和仿真中得到乐趣，激发你去提出问题、建立模型、仿真研究结果。

使用 Simulink 可以使你超越理想的线性模型，建立更真实的线性模型，比如考虑摩擦力、空气阻力、齿轮滑动以及一些描述现实世界的现象。它使你的计算机成为一个模拟分析各种无法实验或不存在系统的实验室，像汽车离合器系统、机翼扰动系统、生态系统、货币系统等。

Simulink 是一个非常实用的系统建模仿真系统，世界上有成千上万的工程师用它来模拟和解决现实问题，它同样会在你的职业生涯中助你一臂之力。

Simulink6.0 实时工作环境为用户提供了仿真过程中可能用到的各种功能的图形化界面支持，包括模型库浏览器、模型窗口、调试口、可视化结果、对结果的可视化分析等，使仿真工作变得非常方便、易学。

Simulink6.0 提供了非常强大的模型库，包括非常丰富的模型，这些模型使你的建模仿真工作变得更加简单。你可以通过模型浏览器查看其中的所有模型。常用的 Simulink Blockset 有：

- ✓ CDMA Reference Blockset（CDMA 通信系统设计与分析）；
- ✓ Communication Blockset（通信系统工具箱）；
- ✓ Dial&Gauges Blockset（交互式图形和控制面板设计工具箱）；
- ✓ DSP Blockset（数字信号处理工具箱）；
- ✓ Fixed-Point Blockset（定点运算控制系统工具箱）；
- ✓ Motorola DSP Developer's Kit（Motorola DSP 开发工具）；
- ✓ Nonlinear Control Design Blockset（非线性控制设计工具箱）；
- ✓ Sim Power System（电力电动工具箱）；
- ✓ TI DSP Developer's Kit（TI DSP 开发工具）；
- ✓ Sim Mechanics（机构仿真）；
- ✓ Neural Network Blockset（神经网络工具箱）；
- ✓ Stateflow（流程控制）；
- ✓ Real-Time Workshop（实时系统）。

Simulink6.0 的模型窗口可以为你提供极其方便的所见即所得的建模环境。只需要轻点几下鼠标，就可以建立一个模型，验证你的实验或构想。详细的使用方法将在下面专门介绍。

Simulink6.0 同样提供了诊断和调试模型的可视化工具，让你方便地对建立的模型进行测试、调整，最终实现预定的目标。

2.1.6 Simulink6.0 库浏览器界面

打开 MATLAB，在命令行中运行 Simulink 或单击 MATLAB 工具栏中的 Simulink 图标，将打开 Simulink Library Browser，如图 2-2 所示，或者在 MATLAB 命令行中运行 Simulink3，将会弹出如图 2-3 所示的窗口，这是传统的 Simulink 模型库显示方式。

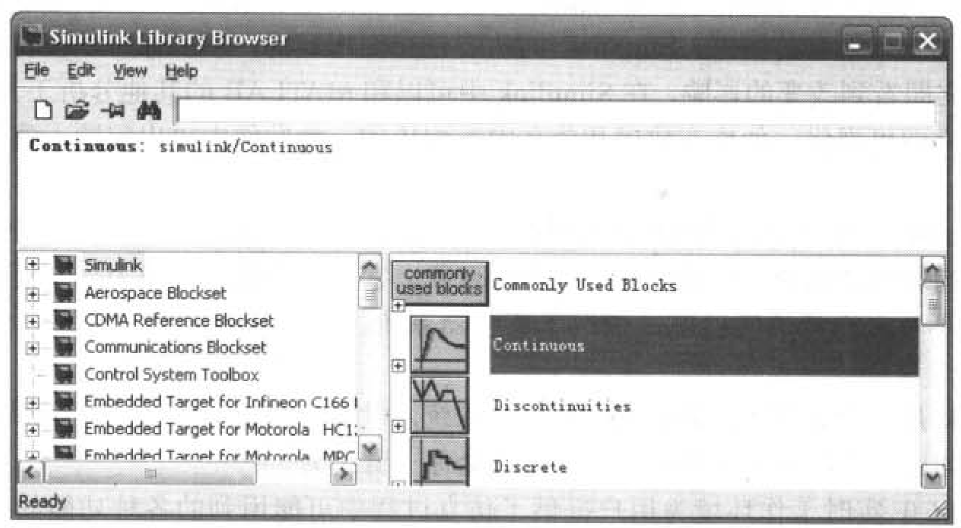


图 2-2　Simulink Library Browser 界面

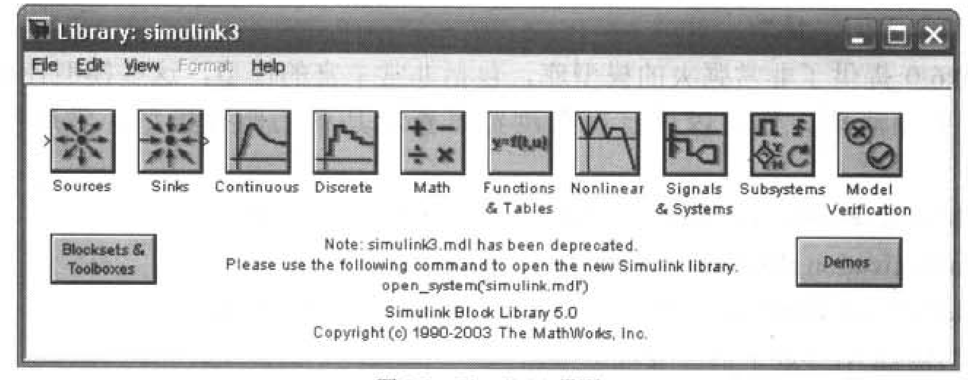


图 2-3　Simulink3 界面

模型库为用户提供了非常丰富的模块组，主要包括：Simulink 基本模块、Aerospace Blockset、Fuzzy Logic Toolbox、Real Time Workshop、Sim Mechanics、Sim Power System、Virtual Reality Toolbox 和 Stateflow 等。

为了方便介绍浏览器中的各个 Simulink 模块组，可用另外一些方式访问这些模块组，这样可以很好地显示模块组的全貌。操作方法：双击相应的模块组名或者单击模块组名称前的加号，则在右侧显示该模块组内的所有模块，如图 2-4 所示，或者右击相应的模块组名称，从弹出的菜单中选择 Open the Continuous Library 命令，如图 2-5 所示，弹出一个新窗口，如图 2-6 所示，其中只显示相应的模块组。

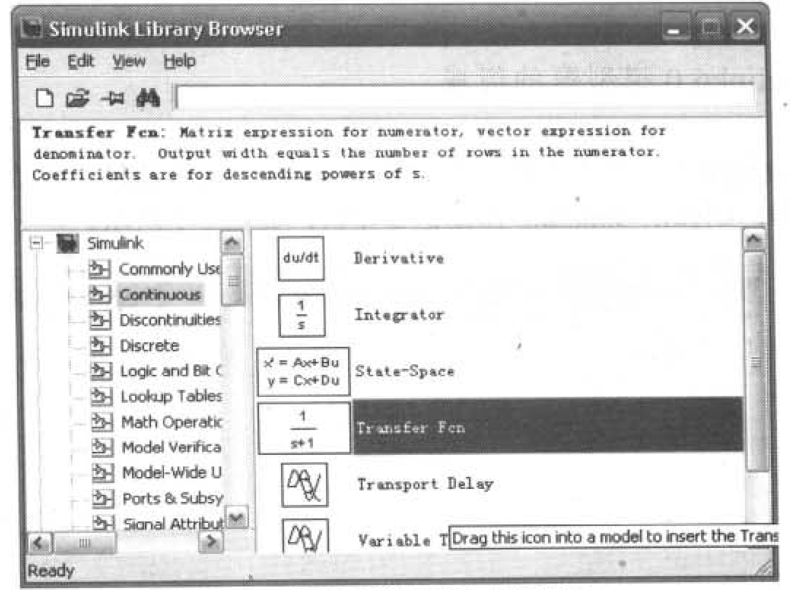


图 2-4　右侧显示该模块组内的所有模块

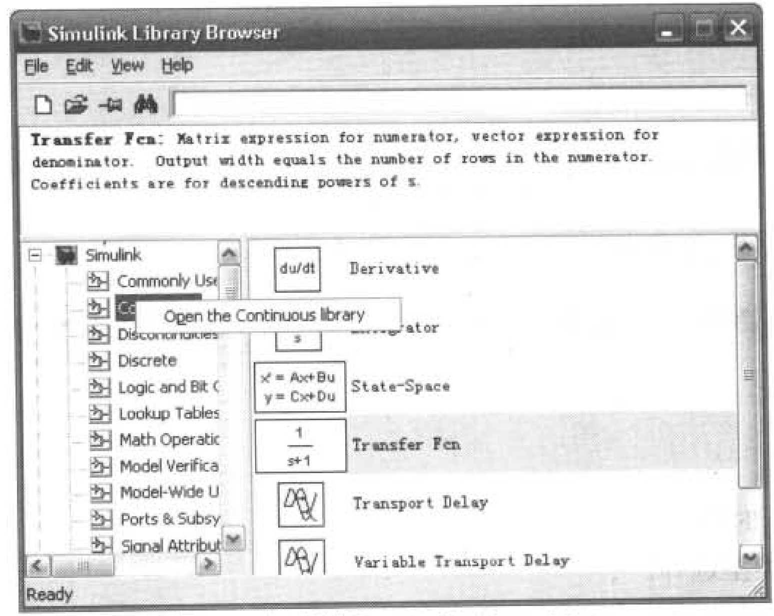


图 2-5　右击菜单 Open the Continuous Library

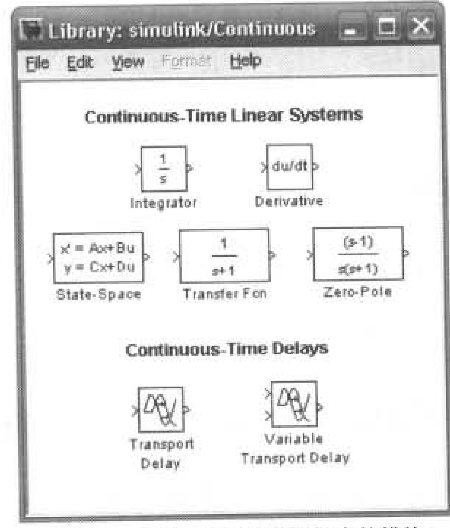


图 2-6　新窗口中显示模块组中的模块

2.1.7 Simulink6.0 模型窗的组成

单击模型库浏览器工具栏上的新建按钮或从其 File 菜单选择新建或者打开一个现有的 Simulink 仿真模型，则弹出 Simulink 建模仿真窗口，如图 2-7 所示。

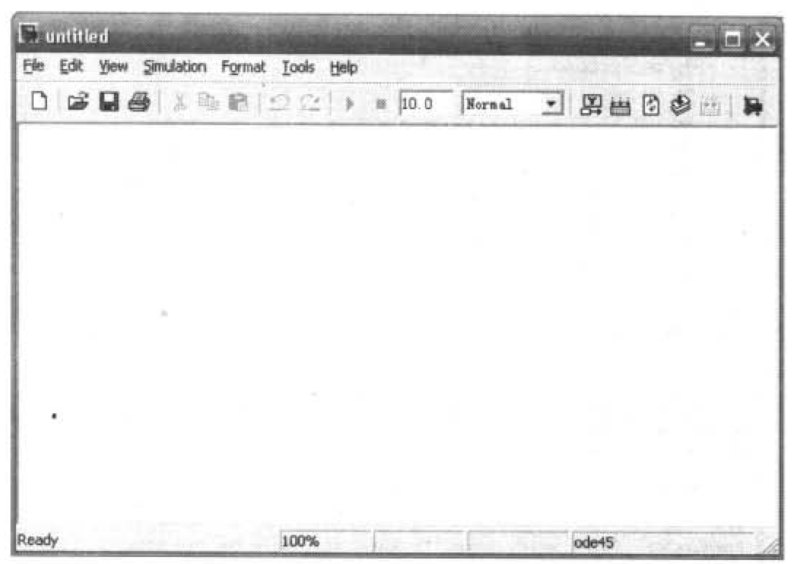


图 2-7 Simulink 建模仿真窗口

其中工具栏如图 2-8 所示。



图 2-8 工具栏

▸开始仿真。

▪停止仿真。

10.0 仿真终止时间。

Normal 仿真类型选择。

当鼠标指针位于模块上方时显示输出值。

增量构建。

刷新模块。

更新标签。

构建子系统。

显示 Library Browser 窗口。

启动 Model Explorer。

显示 Model Browser。

回到父系统。

调试模块。

启动仿真后，图标▸变成了图标⏸，图标▪变为▪，表示可用。工具栏如图 2-9 所示。



图 2-9 开始仿真后的工具栏

工具栏基本包括了常用的功能，在菜单中都有对应的命令。例如：单击▣或选择菜单 View|Model Browser Options|Model Browser，都将出现 Model Browser 窗口，如图 2-10 和图 2-11 所示。

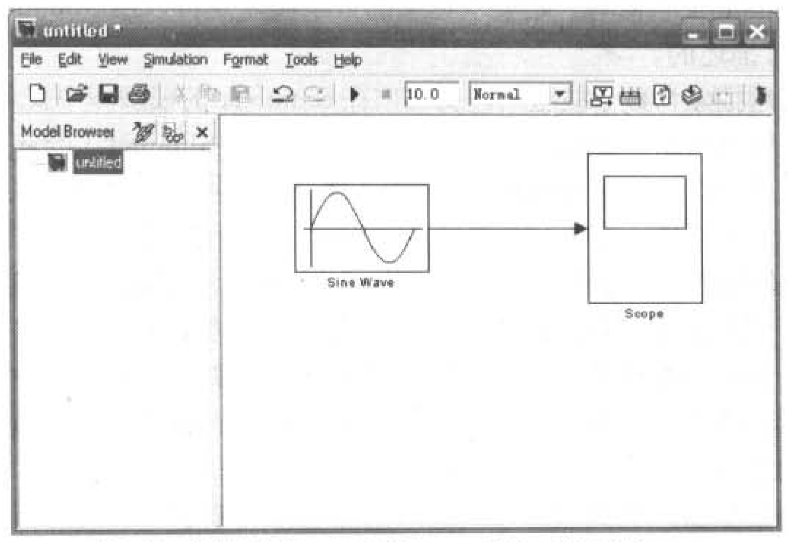


图 2-10　选择菜单 View|Model Browser Options|Model Browser

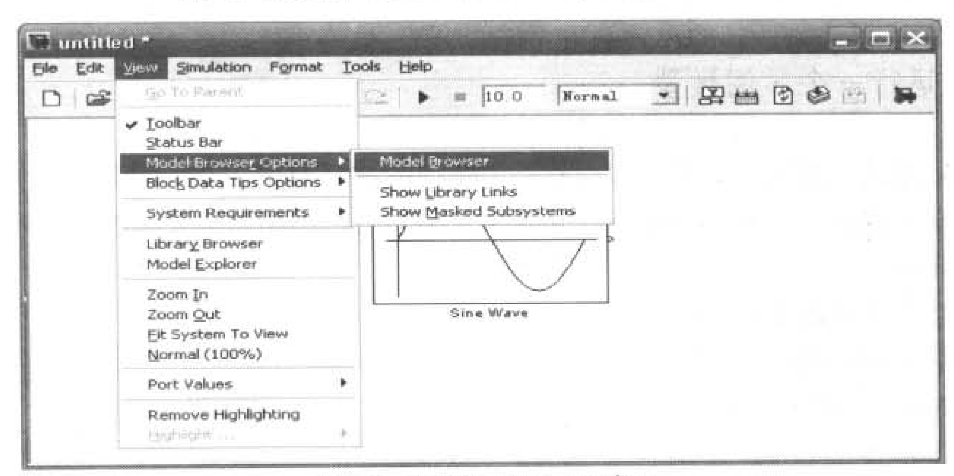


图 2-11　左侧 Model Browser 窗口

菜单命令提供了更多、更强大的功能，在此不一一介绍，在后续的学习中，我们将逐渐认识并学会使用它们。

2.2　Simulink6.0 快速入门

本节将详细介绍 Simulink 的操作，主要包括对模块和连线的操作，介绍如何建立模型，最后运行一个实例模型作为对介绍内容的实践。

2.2.1　建立模型的一般步骤

建立模型是系统仿真的第一步，模型建立的好坏直接影响到后续的许多工作，建立模型的一般步骤如下：

（1）对实际问题或自己的构想进行合理的简化和抽象，使其成为一个可解的数学模型，

简化要做到恰到好处，这就要求建模者对实际问题有深刻的理解，能够把握主要因素，去除次要因素，还要求建模者有一定的数学功底，能把简化后的实际问题或构想抽象成为合适的数学模型。在用 Simulink 仿真时，则要求抽象的数学模型是可以用 Simulink 的自带模型库或 MATLAB 语言进行描述的。

（2）把实际问题抽象为数学模型后，就可以在 Simulink 的模型库中找到对应的基本模块，然后把它们添加到模型窗口。添加操作非常简单，只需要从 Simulink Library Browser 中拖动需要的模块到模型窗口中即可。拖动到模型窗口后，下一步就是按照数据流关系把各个模块进行连线操作，为了增加可读性，还要对各个模块、连线、模型进行注释说明，对模块进行编辑、设置。对模块的编辑操作、对信号线的编辑操作、对模型的注释将在第 3 章作详细的介绍。

（3）Simulink 模型搭建好以后就可以进行仿真，任何模型尤其是复杂的模型不可能没有任何 Bug，或者一次就完全成功，与验证数据完全吻合。所以仿真过程中需要不断修改各个模块的属性、仿真的属性，甚至修改模型本身，以使仿真能够尽量真实地反映现实世界。这是一个不断反复的过程。

建立了正确的模型后，我们就可以用它来指导工作实践了。下面运行一个 MATLAB 自带的 Simulink 模型，向读者演示 Simulink 如何对建立好的模型进行仿真。

2.2.2 运行一个示例模型

Simulink 帮助中提供了一些有趣、复杂、实用的演示模型，此处列举一个关于弹球的力学模型，向读者演示一下 Simulink 进行仿真的大概过程。模型的创建和设置会在后面的章节进行详细的介绍。

运行模型的步骤如下：

（1）启动 MATLAB。

（2）在 MATLAB 命令窗中输入：

```
>> bounce
```

此命令会启动 Simulink，直接打开演示程序模型窗口，如图 2-12 所示。

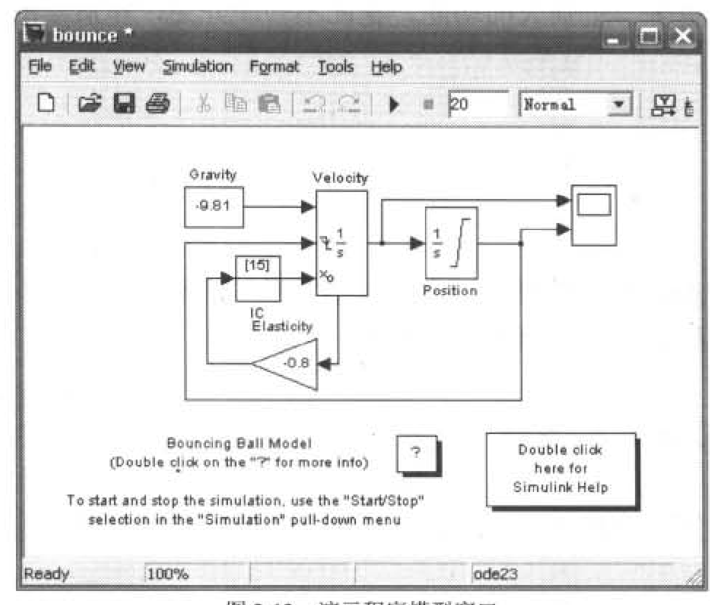


图 2-12 演示程序模型窗口

（3）在第（2）步中打开模型的同时会弹出 bounce 模型的 Scope 窗口，如图 2-13 所示。如果未弹出，也可以通过双击图 2-12 中的 Scope 模块重新打开。

（4）进行仿真。

共有 4 种操作方法：

- ✓ 在 Simulink 中选择菜单命令 Simulation | Start。
- ✓ 按 Ctrl+T 快捷键。
- ✓ 单击图标 ▶。
- ✓ 在 MATLAB 命令窗中输入：

```
>> sim('bounce.mdl')
```

（5）运行后，Simulink 的仿真结果如图 2-14 所示。

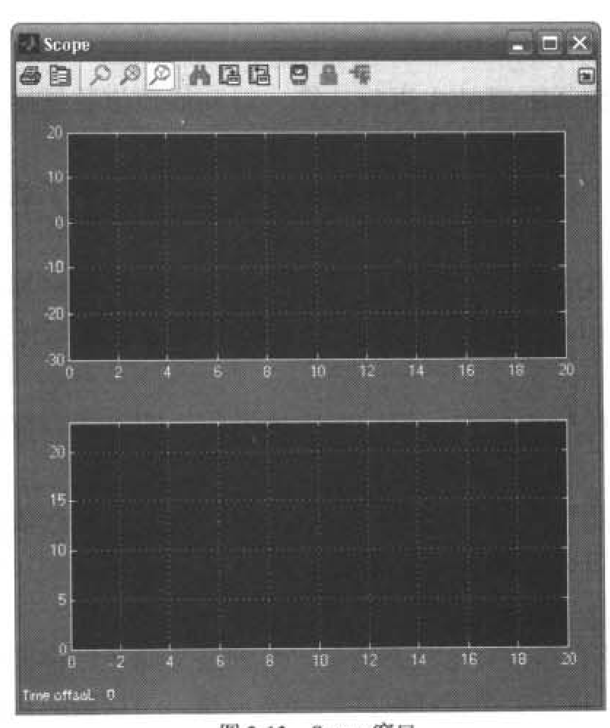


图 2-13　Scope 窗口

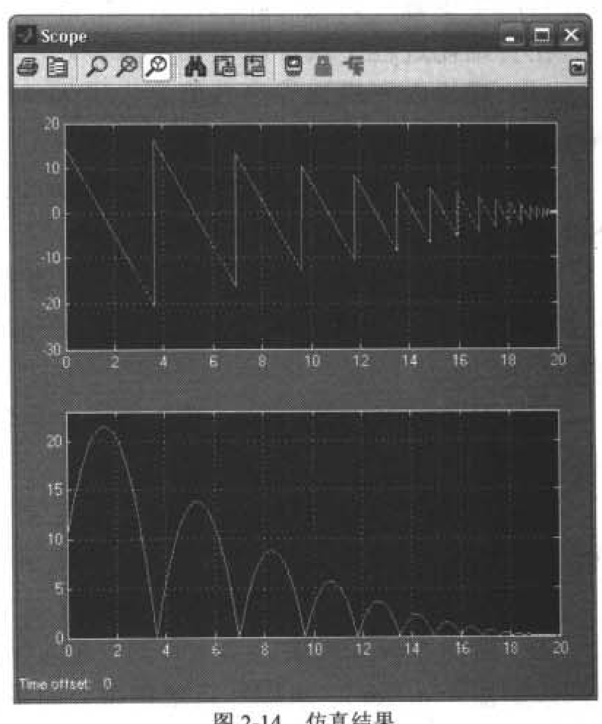


图 2-14　仿真结果

（6）如果需要暂停或结束仿真，方法如下：

- ✓ 在 Simulink 中选择菜单命令 Simulation | Stop。
- ✓ 单击图标 ⏸ 表示暂停，单击图标 ■ 表示结束。

（7）仿真运行结束后，选择菜单命令 File | Close 或直接点右上角的×来关闭程序。

2.2.3　示例的说明

该模型是对一个橡胶球，以初速度 15m/s，从 10m 高度竖直上抛后运动过程的仿真。Scope 上方显示的是橡胶球的速度随时间的变化曲线，Scope 下方显示的是橡胶球的位置随时间变化的曲线。

该模型比较简单，利用高中的物理知识即可理解，在此不作更多介绍，但麻雀虽小五脏俱全，它包含了 Simulink 建模仿真的大部分内容。此模型有信源，有信宿，有积分器，有增益模块等。

2.3 Simulink 是如何工作的

以上介绍了 Simulink 仿真的基本知识，本节将介绍 Simulink 工作的一些基础知识，使读者对 Simulink 的工作原理有一定的了解。

Simulink 模型中的每个模块内都具有一般的特征，包括有一组输入 u、一组输出 y 和一组状态变量 x，如图 2-15 所示。

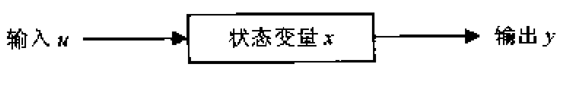


图 2-15 模块示意图

图 2-15 所示的状态变量可以包含有连续状态、离散状态或者是两者的组合。仿真的执行包含有两个步骤：初始化和执行仿真的操作。

在初始化过程中执行以下几个操作。

（1）模块内所设置的参数值会先送到 MATLAB 系统中进行计算，得到的数值数据用来当作以后实际的模块参数设置值。

（2）模型系统中的各个层级（Hierarchy）将被平展（Flattened）开来，每一个子系统（Subsystem）将被相对应的模块所代替。

（3）接下来模块按被处理顺序排列，此时代数回路结构也将被检查出来，此种排序算法产生一个列表，以确保具有代数回路（Algebraic）的模块在驱动输入的模块被更新后才能更新。

（4）检查模块间的连接，是否每一个模块的输出端口与和它所连接的模块输入端口有相同的信号宽度（信号线的数目相同）。

现在可以准备执行仿真操作，仿真是使用数值迭代法求得结果的，每种数值积分法依赖模型提供的连续状态的微分能力。计算微分可以分成两部分来进行。

（1）首先依据排序所确定的次序计算每个模块的输出。

（2）然后再根据模块的当前时刻、输入以及状态来决定状态微分，得到微分向量后再把它送回求解器（Solver），求解器再依据微分向量计算下一个采样时间的状态向量，一旦新的状态向量计算完毕，将更新被采样的来源（例如 Sine Wave 模块）和接受模块（例如 Scope 模块）。

2.3.1 过零点

Simulink 使用过零点技术检测连续信号的间断点。过零点在状态事件的处理和非连续信号的精确积分两种情况下起着重要作用。

1. 状态事件的处理

当状态值的改变使得系统有明显的改变时，系统就经历一个状态事件。状态事件的一个简单例子是碰到地板而反弹的球，当使用变步长求解器仿真这样一个系统时，通常求解器不是采用球正好接触地面的相对应时间步，这样球就好像越过了接触点，似乎球会穿透地面。

Simulink 使用过零点检测，以确保时间步正好（在机器精度范围内）出现在时间状态事件发生的时候，因为时间步正好发生在接触地面的一瞬间，仿真就不会越过接触点，并且速度从负到正的转换非常尖锐（也就是说，在不连续点处没有圆角）。要看反弹球的演示，在

MATLAB 的命令窗口中输入 bounce。

2. 非连续信号积分

数值积分程序是基于这样的假设得到的公式，被积分的信号是连续的并且有连续的导数，如果在某一仿真步遇到了不连续的情况（状态事件）,Simulink 使用过零点检测，以找到不连续是在何时发生的，然后该积分步就积分到不连续点的左边沿，最后，Simulink 跨过不连续点，在信号的下一分段连续的部分开始新的积分步。

表 2-1 所列的模块具有过零点。

表 2-1

模　块	过零点描述
Abs	一个：检测输入信号什么时候以上升或者下降的方向穿过零点
Backlash	两个：一个用来检测什么时候使用上阈值，另一个用来检测什么时候使用下阈值
Dead Zone	两个：一个用来检测什么时候进入死区（输入信号减去下限），另一个用来检测什么时候脱离死区（输入信号减去上限）
Hit Crossing	一个：检测什么时候输入穿过阈值。这一过零点不受 SimulationParameters 对话框中 Disable zero crossing detection 复选框的影响
Integrator	如果存在复位端口，检测什么时候复位发生。如果输出受限制，有三个过零点：一个检测什么时候达到上饱和限，一个检测什么时候达到下饱和限，另一个检测什么时候脱离饱和
MinMax	一个：对于输出向量的每一个元素，检测输入信号什么时候是新的最小值或者最大值
Relay	一个：如果继电器处于关的状态，检测打开的时刻。如果继电器处于开的状态，检测关掉的时刻
Relational Operator	一个：检测什么时候输出改变
Saturation	两个：一个检测什么时候达到或者离开上限，另一个检测什么时刻达到或者离开下限
Sign	一个：检测输入什么时候穿过 0
Step	一个：检测阶跃的时间
Subsystem	对于条件执行的子系统：1 个提供给激活端口（如果存在），一个提供给触发端口（如果存在）
Switch	一个：检测什么时候转换条件发生。

2.3.2　代数回路

代数回路发生于两个或多个模块在输入端口具有信号直接传递（Direct Feed through）而形成反馈回路的情况时，直接传递的模块在不知道输入端口的值的情况下无法计算出输出端口的值，也就是现在时刻的输出是依赖现在时刻的输入值来计算的。当这种情况发生时，Simulink 会在每一次迭代演算完成时去决定它是否会有解。代数回路会减缓仿真执行的速度并且可能会没有解，尽可能避免使用代数回路结构，而使用信号直接传递的模块来构建系统方块模型。

具有此性质的模块有：

- ✓ Gain 模块。
- ✓ 大部分的非线性模块（如 Look-Up Table 模块、Rate Limiter 模块）。
- ✓ 具有相同阶数的分子分母多项式的 Transfer Fcn 模块。
- ✓ 有相同零极点数的 Zero-Pole 模块。

✓ 具有非零 D 矩阵的 State-Space 模块。

图 2-16 所示的是具有代数回路结构系统的例子，此回路包含有 Sum、Transfer Fcn 和 Gain 等模块。

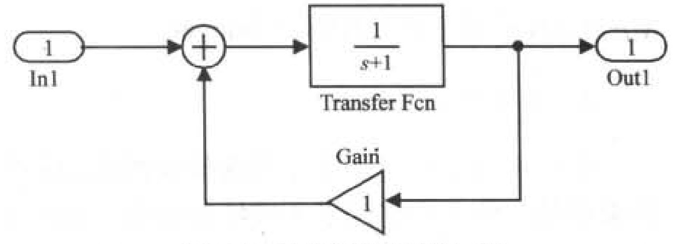


图 2-16 具有代数回路结构系统

如果在 200 个迭代步骤完成后仍无法解得代数回路的解，则 Simulink 会输出错误信息。要切断代数回路，可以在代数回路的任何两个模块间插入一个 Memory 模块。Memory 模块具有延迟积分的功能，常用来切断代数回路。

2.3.3 非代数直接馈通回路

与所有直接馈通回路是代数的一般情况不同，也有一些例外的情况。

（1）回路包含触发子系统。

（2）回路从输出连接积分器的复位端口。

在触发子系统中，求解器可以安全地假设子系统的输入在触发时是稳定的。这就允许使用前一时间步的输出来计算当前时间步的输入，因此排除了代数回路求解器的需要，如图 2-17 所示。触发子系统如图 2-18 所示。

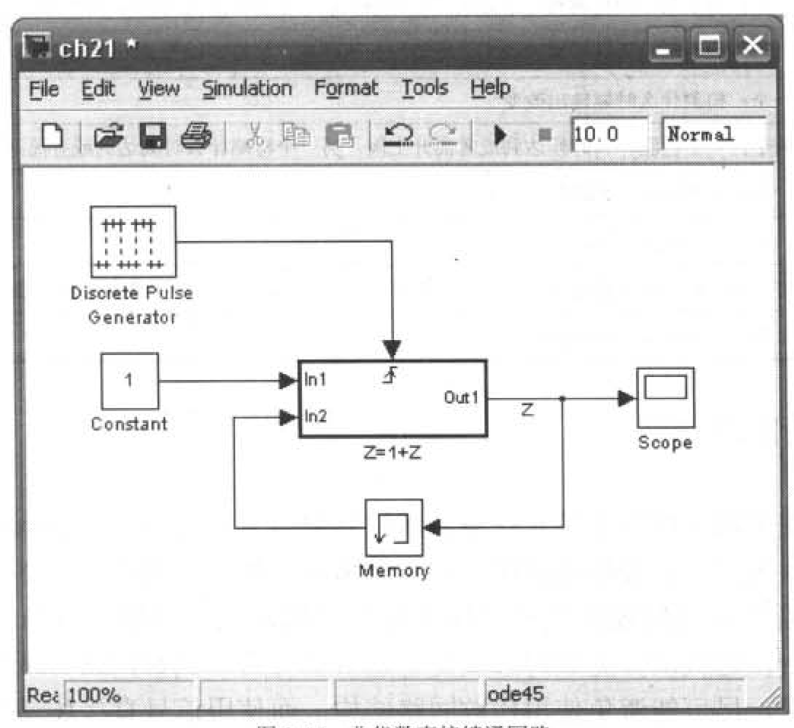


图 2-17 非代数直接馈通回路

该系统可以有效地求解方程式：z=1+u，其中 u 是子系统触发最后的 z 值。系统的输出在系统显示器上显示为阶梯函数，如图 2-19 所示。那么，在该示例中如果去除系统中的触发器和 Memory 模块，如图 2-20 和图 2-21 所示，结果又会怎样呢？此时，每个时间步加法器子系统 u2 端口的输入等于当前步子系统的输出，而系统的数学表达式为 z=z+1，表明没有合法数学公式。故报错，如图 2-22 所示。

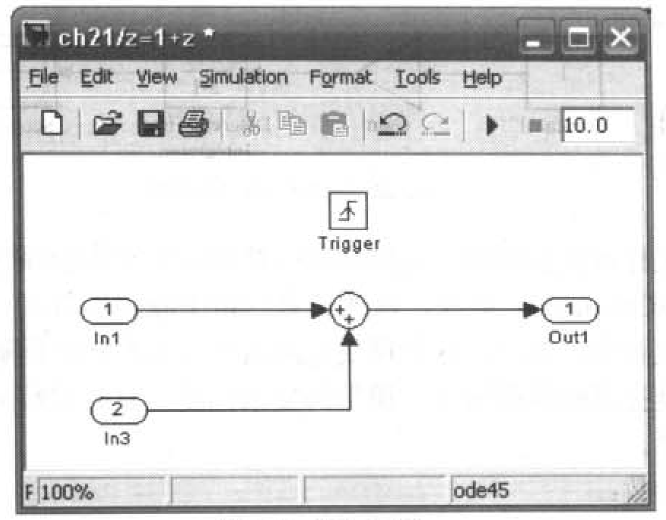



图 2-18 触发子系统

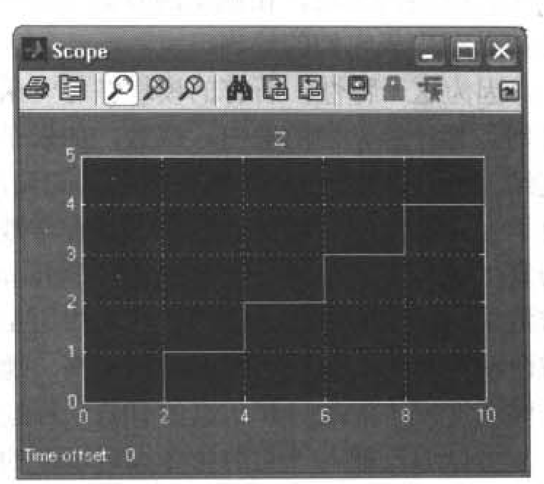


图 2-19 输出阶梯函数

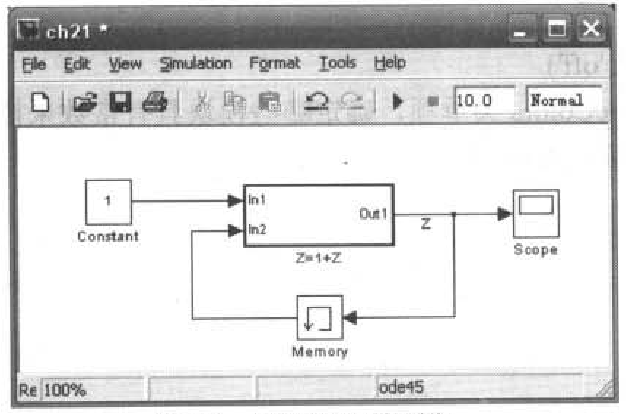


图 2-20 去除触发器后的系统

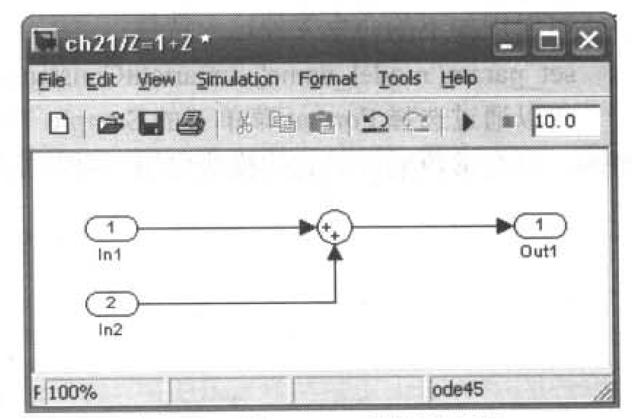


图 2-21 去除 Memory 模块后的系统

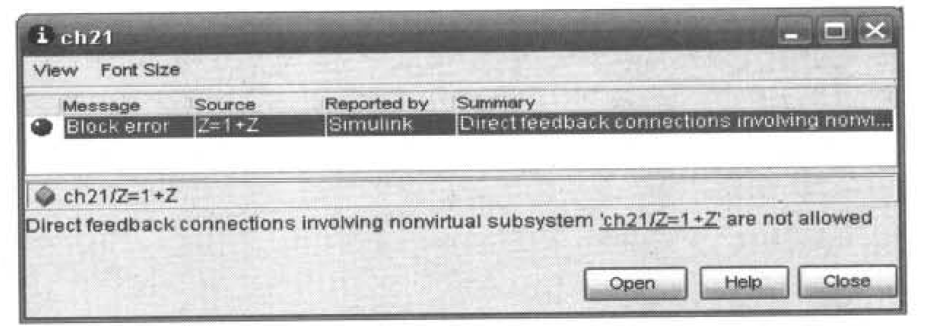


图 2-22 错误信息

2.3.4 不变的常量

模块要么具有明确指定的采样时间，要么从馈送它们的模块（或由它们馈送的模块）那儿继承采样时间。

Simulink 指定 Constant 模块的采样时间是无穷大，也被称作常数采样时间(constant sample time)。如果另外的模块从 Constant 模块那里接收输入，它们将具有常数采样时间，而不从另外的模块那里继承采样时间。这就意味着这些模块的输出在仿真期间不会改变，除非模型的用户明确改动它们。例如，在如图 2-23 所示的模型中，Constant 模块和 Gain 模块都具有常数采样时间。因为 Simulink 支持在仿真期间改变模块参数的能力，所有模块甚至包括具有常

数采样时间的模块，在模型的有效采样时间点都必须生成它们的输出。

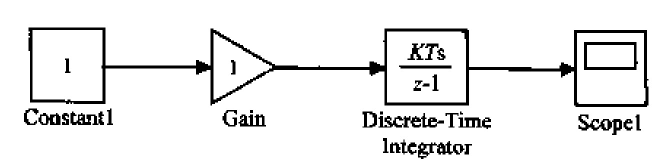


图 2-23 不变常量输入的模型

因为这一特性，所有的模块在每一采样时间点都计算它们的输出，或者，如果是纯连续系统，在每一仿真步都计算它们的输出。对于在仿真期间参数不会改变的具有常数采样时间的模块，在仿真期间计算这些模块会减慢仿真的速度。可以设定不变常量（InvariantConstants）参数，以从仿真回路中除去所有具有常数采样时间的模块。这一特性的作用是双重的：第一，这些模块的参数在仿真期间不能够被改动；第二，仿真速度会得到提高。速度提高的程度取决于模型的复杂程度，具有常数采样时间的模块的数目和仿真的有效采样速率。

可以通过输入下面的命令设定模型的参数：

set_param('model_name', 'InvariantConstants', 'On')

可以用如下的命令关掉这一特性：

set_param('model_name', 'InvariantConstants', 'off')

可以通过选择 Format 菜单下的 Sample TimeColors 菜单项，确定哪些模块具有常数采样时间，具有常数采样时间的模块可用一种特定颜色来显示。

第 3 章　模型的建立

前面几章已经向读者介绍了 Simulink 建模仿真的大致过程，本章将向读者详细介绍模型的相关内容。

本章主要内容：

➢ 模型的概念和文件操作
➢ 模块的操作
➢ 常用的模型库
➢ 仿真的配置

3.1　模型的概念和文件操作

Simulink 仿真的对象即 Simulink 模型，本节将向读者介绍 Simulink 模型的概念和文件操作的相关知识。

3.1.1　什么是 Simulink 模型

Simulink 是一个用来对动态系统进行建模、仿真和分析的软件包。它支持线性和非线性系统，连续和离散时间模型，或者是两者的混合，系统还可以是多采样率的，比如系统的不同部分拥有不同的采样率。对于建模，Simulink 提供了一个图形化的用户界面（GUI），可以用鼠标点击和拖拉模块的图标建模，通过图形界面，可以像用铅笔在纸上画图一样画模型图，这是以前需要用编程语言明确地用公式表达微分方程的仿真软件包所远远不能相比的。Simulink 包括一个复杂的由接收器、信号源、线性和非线性组件以及连接件组成的模块库，当然也可以定制或者创建用户自己的模块。

所有模型是分级的，因此可以通过自上而下或者自下而上的方法建立模型，可以在最高层面上查看一个系统，然后通过双击系统中的各个模块进入到系统的低一级层面以查看到模型的更多细节，这一方法提供了一个了解模型是如何组成以及它的各个部分是如何相互联系的方法。

定义完一个模型以后，就可以通过 Simulink 的菜单或者在 MATLAB 的命令窗口输入命令对它进行仿真，菜单对于交互式工作非常方便。而命令行方式对于处理成批的仿真比较有用（例如，你在进行 Monte Carlo 仿真时想使参数遍历某一范围的值），使用 Scopes 或者其他的显示模块，可以在运行仿真时观察到仿真的结果。另外，还可以在仿真时改变参数并且立即就可看到有什么变化，仿真的结果可以放在 MATLAB 的工作空间（workspace）中以待进一步的处理或者可视化。

模型分析可使用的工具包括可直接通过命令行方式调用的线性化和整理（trimming）工具，MATLAB 的其他各种工具，以及所有应用程序工具箱。因为 MATLAB 和 Simulink 是集

成在一起的，所以用户可以在任何环境的任意点对用户的模型进行仿真、分析或修改。

3.1.2 模型文件的操作

Simulink 中的各个模型都是可以适当修改的，不过这需要用户对 Simulink 有非常深入的了解。Simulink 的模型文件的后缀为 mdl，其实可以显示为一组代码，主要由以下几个部分组成。

✓ Model section：定义模型的参数。

✓ BlockDefaults section：模块的默认设置。

✓ AnnotationDefaults sections：模型注解的默认值。

✓ System section：顶层系统或者子系统的描述参数，包括 line、block 和 annotation 等。

下面通过一个简单实例来看看具体代码文件的内容。通过下面这个例子就可以发现 MATLAB 和 Simulink 这两种模型形式的不同，最后会选用 Simulink 窗口建模。

在 MATLAB 命令窗口中输入：

```
>>edit demo.mdl
```

则会在 M 文件编辑框中出现如下代码，代码的具体含义可以参照模型来理解。这些代码就是以 M 文件形式打开的 demo.mdl 文件，

```
Model {
  <Model Parameter Name> <Model Parameter Value>
  ...
  BlockDefaults {
    <Block Parameter Name> <Block Parameter Value>
    ...
  }
  AnnotationDefaults {
    <Annotation Parameter Name> <Annotation Parameter Value>
    ...
  }
  System {
    <System Parameter Name> <System Parameter Value>
    ...
    Block {
      <Block Parameter Name> <Block Parameter Value>
      ...
    }
    Line {
      <Line Parameter Name> <Line Parameter Value>
      ...
      Branch {
        <Branch Parameter Name> <Branch Parameter Value>
```

```
        ...
      }
    }
    Annotation {
      <Annotation Parameter Name> <Annotation Parameter Value>
      ...
    }
  }
}
```

The model file consists of sections that describe different model components:

The Model section defines model parameters.

The BlockDefaults section contains default settings for blocks in the model.

The AnnotationDefaults section contains default settings for annotations in the model.

The System section contains parameters that describe each system (including the top-level system and each subsystem) in the model. Each System section contains block, line, and annotation descriptions.

3.2 模块的操作

Simulink 仿真的基础是模块，本节主要介绍模块的操作，包括模块的基本操作、向量化模块和标量扩展以及模块参数的设置。

3.2.1 模块的基本操作

下面介绍模块的操作。

1. 调整模块大小

通过调整模块大小，能够直接清晰地看到模型的参数，提高模型的可读性。有些模块，如 Gain 增益模块，当参数位数太大时就用字母代替，此时可以适当地扩大模块的大小，使之显示所设置的参数。调整模块的操作步骤如下：

- 新建一个模型窗口，命名为 Size.mdl。
- 选择 Source 中的 Constant 模块库，如图 3-1 所示，并将其拖动到模型窗口，双击此模块，设置其 Constant value 为 123456789，如图 3-2 所示，由于数字太长，无法在图标中显示，故显示为“-C-”，如图 3-3 所示。
- 为了能够显示常数，可以扩大模块，单击 Constant 模块，然后用鼠标指针放在 4 个黑方块上，此时鼠标指针会改变形状，然后拖动鼠标，调整模块大小，则可以显示常数的所有位，如图 3-4 所示。

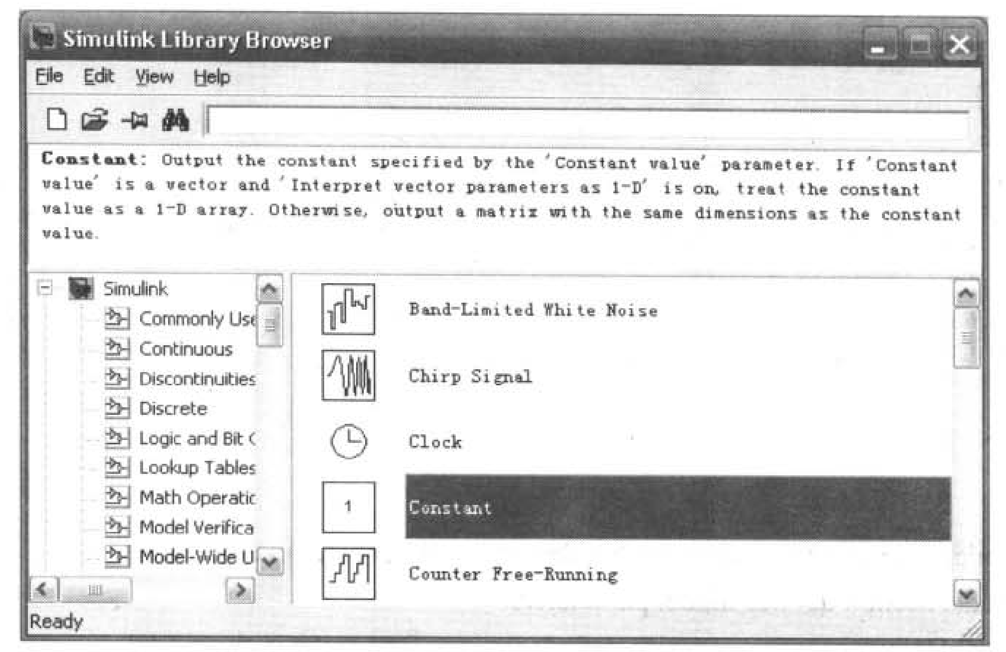


图 3-1 Constant 模块库

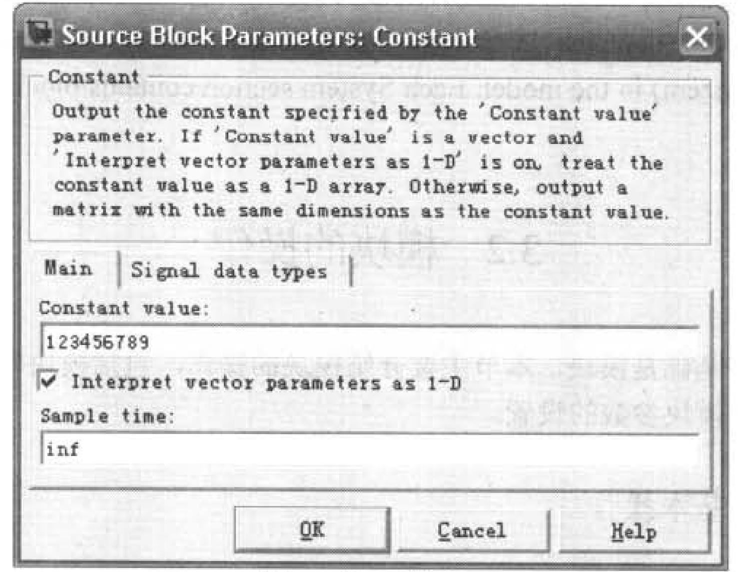


图 3-2 设置参数窗口

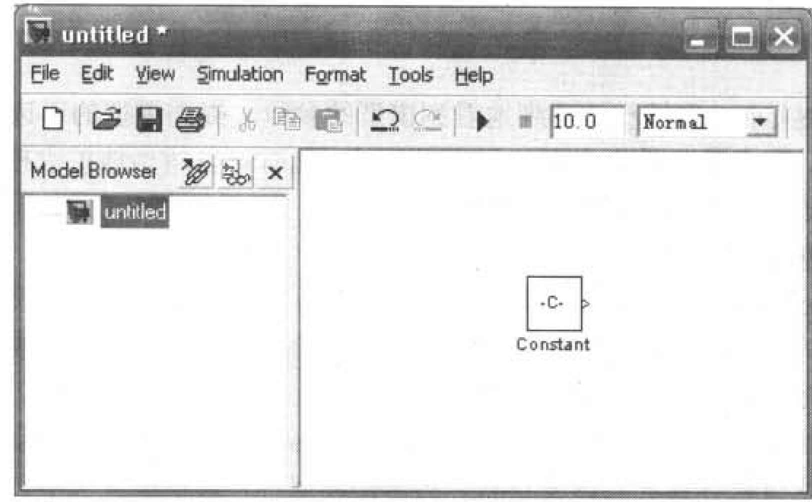


图 3-3 数字太长无法在图标中显示

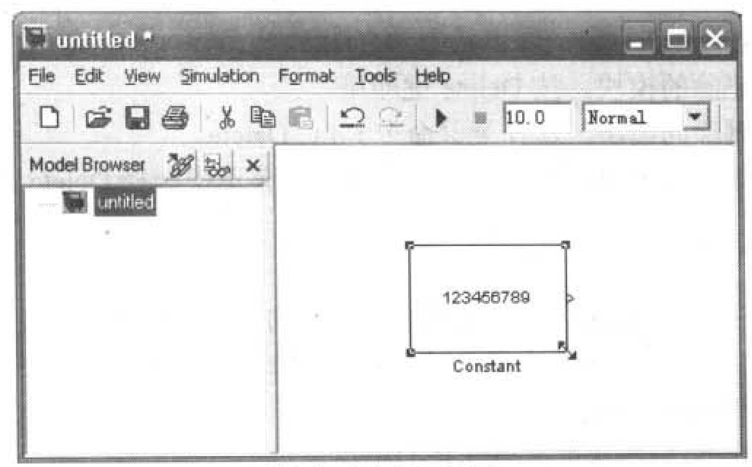


图 3-4　放大后正常显示

2. 模块旋转

- 单击选中要旋转的模块，选择菜单命令 Format | Rotate block，如图 3-5 所示。

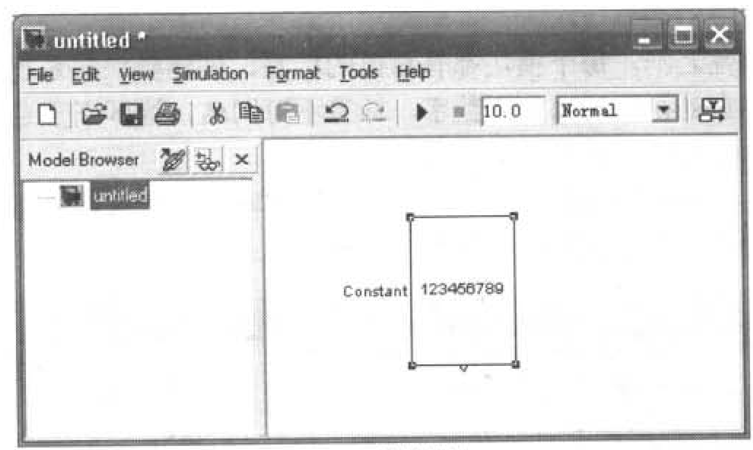


图 3-5　旋转模块

- 单击选中要旋转的模块并右击，在右键菜单中选择 Format | Rotate block 命令。

Rotate block 是顺时针旋转 90°，Flip block 是旋转 180°，用法同上。

3. 模块复制

在建模过程中经常遇到大量功能重复和设置相同的模块，如果每个都从模型库中拖过来，然后进行参数设置，则操作效率低下，且容易出错，而在这种情况下复制的效果可能更好。以下是几种复制的方法。

- 单击选中要复制的模块，选择菜单命令 Edit | Copy，然后选择菜单命令 Edit | Paste。
- 单击选中要复制的模块，用快捷键 Ctrl+C 复制，然后用 Ctrl+V 粘贴。
- 单击选中要复制的模块，按住 Ctrl 键，然后用鼠标拖动该模块即可。
- 按住鼠标右键拖动要复制的模块，这种方法最为方便，推荐使用。

4. 模块删除

当模块中出现了多余的模块，即使不删除，Simulink 也可以正常运行，并不会因此而影响仿真结果。但多余的模块会降低模型的可读性，并会在 MATLAB 命令窗口中出现大量的

警告信息，对调试程序十分不利。以下是删除方法。

- 单击选中要删除的模块，按 Delete 键即可。
- 单击选中要删除的模块，选择菜单命令 Edit | Delete。
- 单击选中要删除的模块，然后右击，在弹出的菜单中选择 Delete 命令。

5. 模块选择

在建模过程中，有时需要选择多个模块进行同样的操作，如复制、旋转、删除、移动等，在进行这些操作之前，可以一次性选择需进行相同操作的所有模块，统一操作，加快操作速度。以下是选择多个模块的方法。

- 按住 Shift 键，然后依次单击要选择的模块。
- 使用框选，按下鼠标左键或右键均可，拖动鼠标画出一个矩形框，框出要选择的模块即可。

在两种方法中，前者适合选择零散的模块，后者适合选择相邻的一个区域内的模块。

6. 标签设置

标签是模块的属性之一，每个模块都有自己的标签，创建模块式系统会自动命名。对于相同的模块，系统会自动在它后面加上数字。标签不可同名，这区别于连线标签，很多情况下需要修改模块的标签来提供系统或模块的可读性。

修改标签方法：在所要修改的标签上面单击，标签则呈现可编程状态。如图 3-6 所示，输入想要的标签，设置完成后在空白处单击，修改完成。

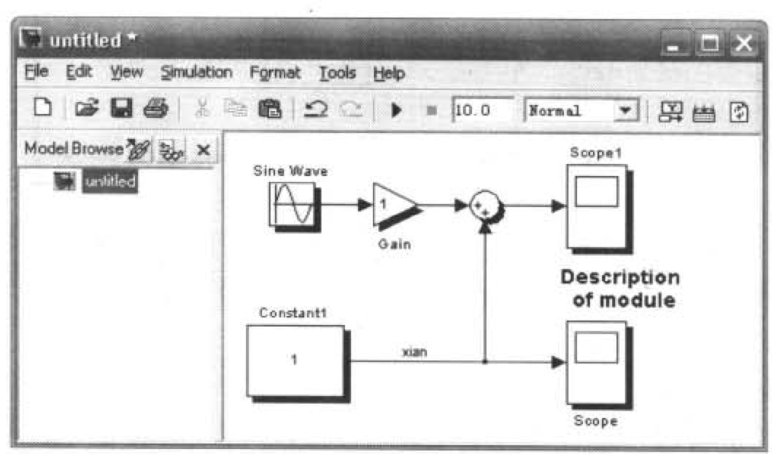


图 3-6 模块阴影

修改标签位置的方法：

- 单击所要编辑的模块，然后选择菜单命令 Format | Flip name。
- 右击所要编辑的模块，从弹出的菜单中选择 Format | Flip name 命令。

隐藏标签方法：

- 单击所要编辑的模块，然后选择菜单命令 Format | Hide name。
- 右击所要编辑的模块，从弹出的右键菜单中选择 Format | Hide name。

显示标签的方法：

- 单击所要编辑的模块，然后选择菜单命令 Format | Show name。
- 右击所要编辑的模块，从弹出的右键菜单中选择 Format | Show name。

7. 模块阴影

有时为了提高系统的可读性，或者出于强调模型中的重点模块等，可以通过为模块增加阴影来突现模块，能够增强视觉效果，有助于理解模型系统，操作方法如下：

- 单击所要编辑的模块，然后选择菜单命令 Format | Show Drop shadow。
- 右击所要编辑的模块，从弹出的右键菜单中选择 Format | Show Drop shadow。

3.2.2 向量化模块和标量扩展

几乎所有的 Simulink 模块都接受标量或向量输入，产生标量或向量输出，并且允许用户提供标量或向量参数，这样的模块将在本书中被称之为向量化了的模块。

可以通过选择 Format 菜单下的 Wide Vector Lines 菜单项以确定一个模型的哪些连线传输向量信号，当这一选项被选取时，传输向量信号的连线将比传输标量信号的连线显得粗一些。

如果要在选择 Wide Vector Lines 后改变模型的显示，必须选择 Edit 菜单下的 Update Diagram 菜单项，以更新模型的显示，开始仿真也可以更新模块图的显示。

- 输入和参数的标量扩展

标量扩展是将一个标量值转换为同样元素的向量，Simulink 对大部分模块的输入或参数都可进行标量扩展。

- 输入的标量扩展

当使用有多个输入端的模块（诸如 Sum 或 Relational Operator 模块）时，可以将向量输入和标量输入混合在一起，此时，标量将扩展成相同元素的向量，而宽度与向量输入相等。

如果多个模块的输入是向量，那么它们包含元素的个数应该相等。

如图 3-7 所示，这一模型具有标量输入和向量输入，Constantl 模块的输入将被标量扩展，以匹配 Constant 模块的向量输入。它的输入将被扩展为向量［2 2 2 2 2］。模块输出如图 3-8 所示。

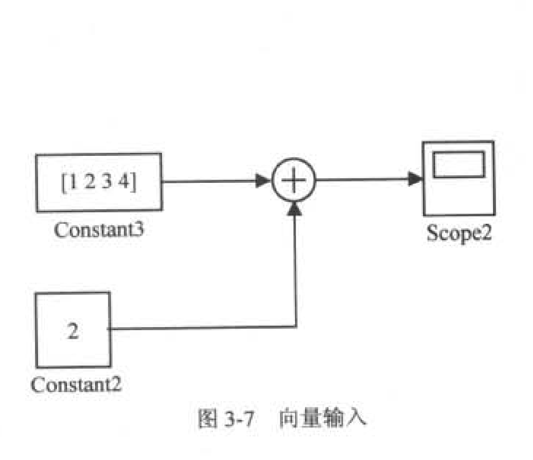


图 3-7 向量输入

图 3-8 模块输出

- 参数的标量扩展

可以指定向量化了的模块的参数为向量或者标量。指定向量参数时，每一个参数元素将与输入向量的相对应的元素相关联。当指定标量元素时，Simulink 将自动地应用标量。

3.2.3 模块参数的设置

双击模块或者右击选择 Constant Parameters…，（见图 3-9）弹出如图 3-10 所示的对话框，不同的模块有不同的界面，根据模块的不同对属性进行设置。

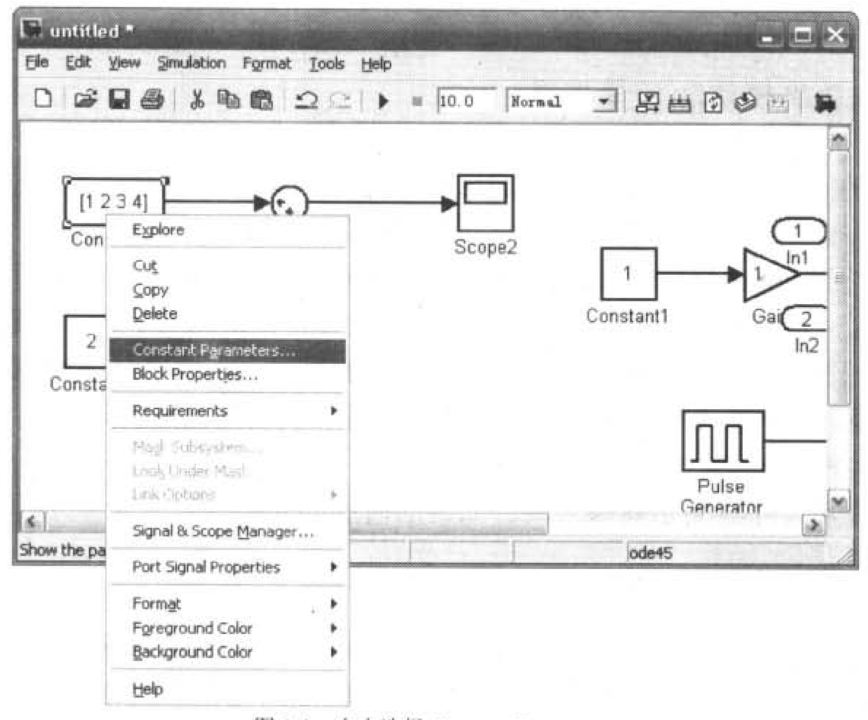


图 3-9　右击选择 Constant Parameters

右击选择 Block Properties…，弹出如图 3-11 所示的对话框，对模块属性进行设置。

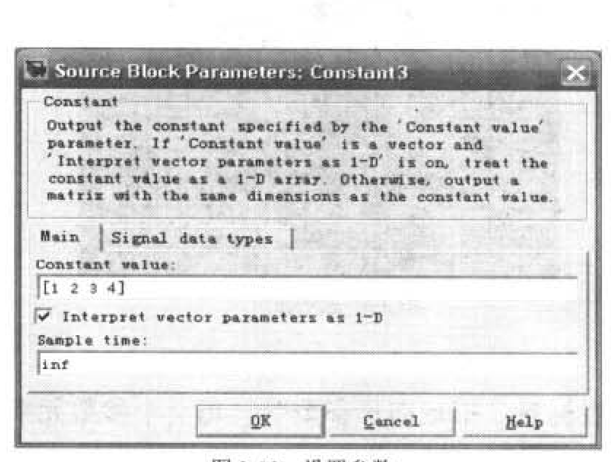


图 3-10　设置参数



图 3-11　模块属性

3.3 信号线的操作

1. 连线绘制

- 将鼠标指针移动到模块输出端，鼠标指针将呈十字形，然后按住鼠标左键，移动到所要连接的模块输入端，在此依次连接各模块即可。
- 单击选中一个模块，按住 Ctrl 键，同时左键单击另一个模块，Simulink 会自动连线两个模块的输出端和输入端。此方法更快捷。
- 将鼠标指针移动到任一条连线上，按住右键不放，并拖动到第三个模块的数输入端，则可使一个输出对应到多个输入。

2. 连线移动

在复杂的模型中，由于有大量的连线，而且连线之间往往容易交叉，严重降低了程序的可读性，因此很有必要移动连线。操作如下：

- 单击希望移动的连线。
- 将鼠标指针移到连线上，鼠标指针形状将变为移动图标（十字形），按住鼠标左键移动到目的位置即可。

3. 节点移动

- 此操作类似于连线移动，只是将鼠标指针放在节点处，此时鼠标指针的形状会变成圆形，再拖放节点到期望的地方即可。

4. 连线删除

同删除模块操作一样有 3 种方法：

- 单击所要删除的连线，然后按 Delete 键。
- 单击所要删除的连线，然后选择菜单命令 Edit | Delete。
- 右击所要删除的连线，在弹出的菜单中选择 Delete。

5. 连线分割

- 选择要编辑的连线，按住 Shift 键，在要分割的地方单击，其形状就会变成圆形，而连线也就在此处被分割成两段。接着就可以拖动新节点到需要的位置，放开节点即可。
- 分割之后可以改变连线的形状。
- 取消分割的方法与分割方法类似，按住 Shift 键，在已经分割的地方单击，分割就会消失。

6. 添加连线标签

添加连线标签，有利于表明连线的功能，标签可以放在连线的任何位置。

在想要添加标签的连线上双击，连线相应的地方会出现一个编辑框。在这个编辑框中输

入标签的文本即可。

7. 编辑连线标签

将鼠标指针放在要编辑的文本上方，然后单击，就呈现可编辑状态，这样就可以编辑标签的内容了。

8. 移动连线标签

将鼠标指针放在要编辑的文本附近（不要放在文本上方，但要在编辑框内），然后单击，标签就会呈被选状态，但还是不能被编辑。

9. 复制连线标签

在建模的过程中可能会遇到重复的标签，如果对每个标签都按部就班地编辑，就显得非常麻烦、费时，且容易出错。而原本存在的连线标签都是固定于某个连线的，不能复制，因此需通过其他方法来解决此问题。此处的标签略不同于前面介绍的标签，不可以被随意移动。

下面介绍 4 种方法来复制连线标签，这 4 种方法都需要在窗口的空白处双击，并在出现的编辑框中输入需要创建的标签，然后按下面的方法复制。

- 单击所要复制的标签，选择菜单命令 Edit | Copy，再选择菜单命令 Edit | Paste。
- 单击所要复制的标签，按 Ctrl+C 组合键，然后按 Ctrl+V 组合键。
- 单击所要复制的标签，按住 Ctrl 键，然后用鼠标拖动要复制的标签。
- 用鼠标的右键拖动要复制的标签，然后将复制的标签拖放到需要的连线上。

备注：此方法复制的标签不能用于下面要介绍的标签传递，实质上只是一个文本。

10. 传递连线标签

连线的标签可以传递，并可以组合和分开，例如模块 Mux 与 Demux、Goto 与 From、Input 和 Output 等。通过连线标签的组合和分开，可以提高连线的可读性。下面举一个例子来演示此操作。

新建一个模型，模块分别来自模块集 Signal Routing，Sink 和 Sources 模块库，建立 sin A 和 sin B 输出连线标签，建立 Mux 和 Demux 模块间的连线标签，双击连线，输入小于号“<”，然后选择菜单命令 Edit | Update Diagram，如图 3-12 所示。在需要传递的任何连线上，只要按照上述方法，都可以得到传递标签。

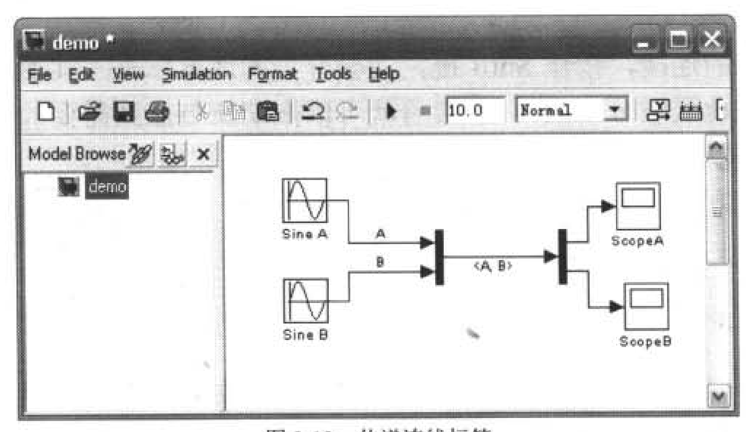


图 3-12 传递连线标签

3.4 对模型的注释

1. 模型说明

使用模型说明可以让模型更加易懂，主要目的就是说明模型的功能和使用方法。对使用Simulink来说，这是非常重要的。下面介绍如何添加模型说明。

在模型窗口中的任何位置双击，此时就出现一个可编辑框，输入需要添加的内容即可。如图3-13所示。

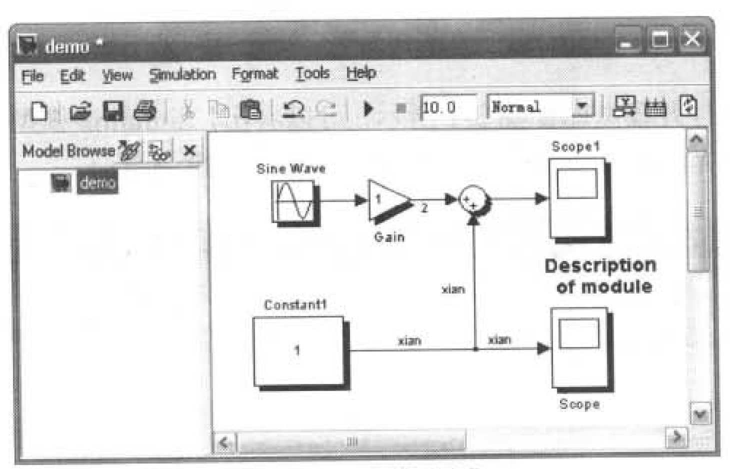


图3-13 添加模型说明

备注：模型说明只支持英文，不能含有汉字，暂时不支持汉字文本。如果带有汉字，在保存此类模型时会弹出一个错误对话框，如图3-14所示，说明不能够保存。

添加完说明后还可以修改字体及其大小，操作如下：

- 选中模型说明。
- 选择 Format | Font 命令，弹出如图3-15所示的对话框。

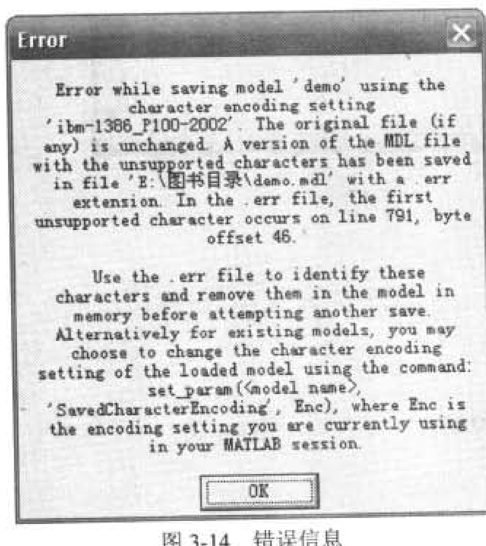


图3-14 错误信息

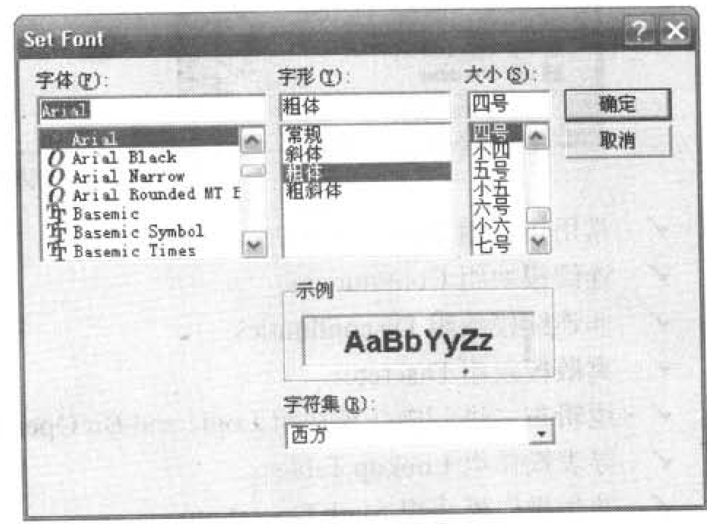


图3-15 设置字体

- 在对话框中进行适当设置后，单击【确定】按钮即可。

2. 模型打印

- 菜单打印，选择 File | Print Setup 命令，设置好各种打印属性，然后选择 File | Print 命令，单击 OK 即可。
- 嵌入文档中打印，主要有两种方法，一是选择 Edit | Copy Model To Clipboard，然后粘贴到 Word、PDF 等其他多媒体文档中即可打印。
- 使用 MATLAB 的 Print 命令可以将图形输出到打印机、剪贴板或其他文档中，详细操作请参阅 MATLAB 帮助文件，或在命令行中运行 Help Print。

3.5 常用的模型库

在 MATLAB 中打开模型库如图 3-1 所示，可以看出，Simulink 模型库包含了丰富的模块组，从图 3-16 中可以看出 Simulink 模型库包含了不同的模块组：

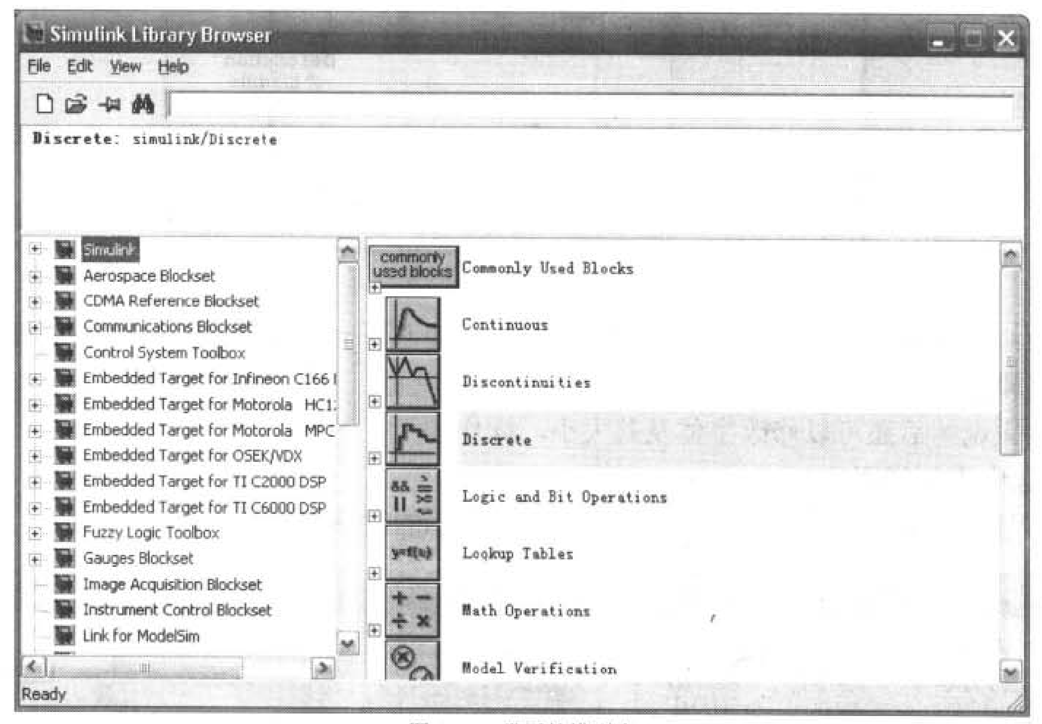


图 3-16 常用的模型库

- ✓ 常用模块组 Commonly Used Blocks；
- ✓ 连续模块组 Continuous；
- ✓ 非连续模块组 Discontinuties；
- ✓ 离散模块组 Discrete；
- ✓ 逻辑与二进制操作模块组 Logic and Bit Operations；
- ✓ 寻表操作组 Lookup Tables；
- ✓ 数学操作模块组 Math Operations；
- ✓ 模型确认操作模块组 Model Verification；
- ✓ Model-Wide Utilities；

✓ 端口与子系统模块组 Ports&Subsystems；
✓ 信号路由模块组 Signal Routing；
✓ 接受器模块组 Sinks；
✓ 信号源模块组 Sources；
✓ 自定义函数模块组 User-Defined Functions；
✓ 附加操作组 Additional Math & Discrete。

此外，用户还可以自定义模块组。

本节只是对 Source 库和 Sink 库做介绍。

3.5.1 Source 库信源

信号源模块组包括常用的信号发生模块，如图 3-17 所示。

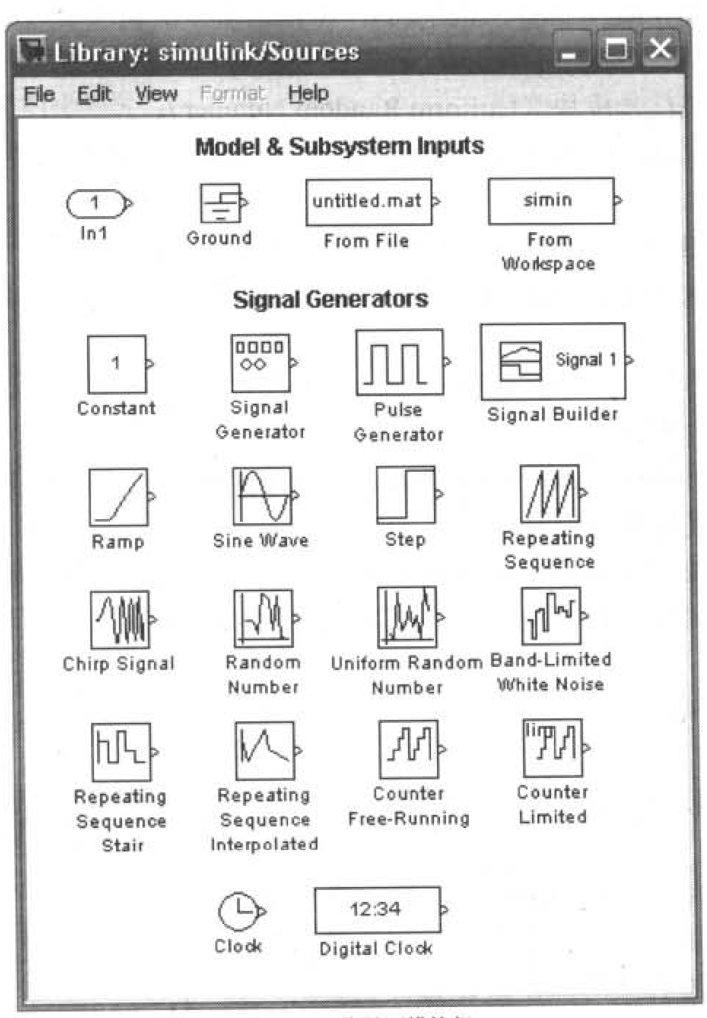


图 3-17　信号源模块组

➢ 输入端口模块（In1）：用来反映整个系统的输入端，在模型线性化与命令行仿真时，这个设置非常有用，可作为信号输入。
➢ 接地模块（Ground）：一般用于表示零输入模块，如果一个模块的输入端没有接其他任何模块，仿真时往往会出现警告，这样可以将该模块接入，功能类似于终结模块（Terminator）。

➢ 从文件中输入数据模块（From File）、从工作区输入数据模块（From Workshop）：从外部输入数据，前者从 .mat 文件中输入，后者从 MATLAB 工作区输入数据。
➢ 常数模块（Constant）：产生不变常数。
➢ 信号发生器模块（Signal Generator）：可产生正弦波、方波、锯齿波等信号，并可以设置幅度和频率。
➢ 脉冲发生器模块（Pulse Generator）：产生脉冲信号，可以设置幅度、周期、宽度等信息。
➢ 信号构造模块（Signal Builder）：在模块窗口双击此模块，在弹出的对话框中绘制信号，即可构造出所需信号。
➢ 斜坡信号模块（Ramp）：产生斜坡信号。
➢ 正弦波信号模块（Sine Wave）：产生正弦波信号。
➢ 阶跃信号模块（Step）：产生阶跃信号。
➢ 重复信号模块（Repeating Sequence）：可构造重复的输入信号。
➢ 变频信号模块（Random Number）：产生正态分布随机信号。
➢ 均匀分布随机信号模块（Uniform Random Number）：产生均匀分布的随机信号。
➢ 限带白噪声（Band-Limited White Noise）：一般用于连续或混合系统的白噪声信号输入。
➢ 时钟模块（Clock）：用于显示和提供仿真时间信号。
➢ 数字时钟模块（Digital Clock）：用于显示在制定的样本间隔内的时间，其他情况保持时间不变。
➢ 重复离散信号模块（Repeating Sequence Stair）：构造可重复输入的离散信号，样本间信号采用零阶保持。
➢ 重复离散信号模块（Repeating Sequence Interpolated）：构造可重复输入的离散信号，样本间信号采用线性插值。
➢ 累加信号模块（Counter Free-Running）：信号不断累加，当累加的信号大于 2^N-1 时，信号会自动回零，其中 N 为参数设置对话框 Number of Bits 所设置。

3.5.2 Sink 库信源

信号接受模块组包括常用的离散模块，如图 3-18 所示。

➢ 输出到动作空间模块（Out1）：用来反映整个系统的输出端，这样的设置在模型线性化与命令行仿真时是必须的，在系统直接仿真时，这样的输出将自动在 MATLAB 工作空间中生成变量。
➢ 终结模块（Terminate）：用来终结输出信号，在仿真的时候可以避免由于某些模块的输出端无连接信号而导致的警告。
➢ 输出数据到文件模块（To File）：将模块输入的

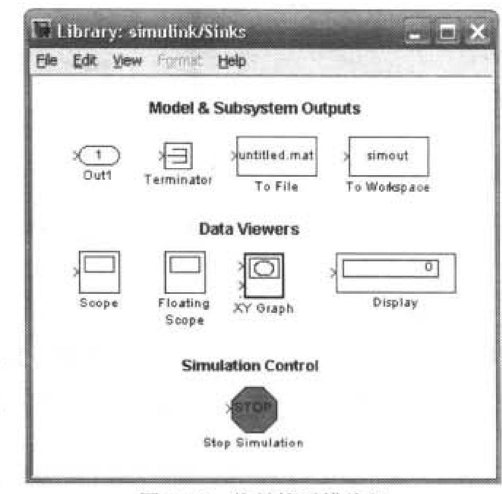


图 3-18　信号接受模块组

数据输出到.mat 文件当中。

- 输出数据到工作区模块（To Workspace）：将模块输入的数据输出到工作区当中。
- 示波器模块（Scope）：将输入信号输入到示波器中显示出来。
- X-Y 示波器模块（XY Graph）：将两路信号分别作为示波器的两个坐标轴，以显示信号的相轨迹。
- 终止仿真模块（Stop Simulation）：如果输入为零，则强制终止仿真。

3.6 仿真的配置

在 Simulink 模型窗口中选择 Simulation | Configuration Parameters 命令，弹出如图 3-19 所示的仿真参数设置对话框。图中左侧列表框中的目录树包括 Solver、Data Import/Export、Optimization、Diagnostics、Hard ware Implementation、Model Referencing 和 Real-Time Workshop 等几项。

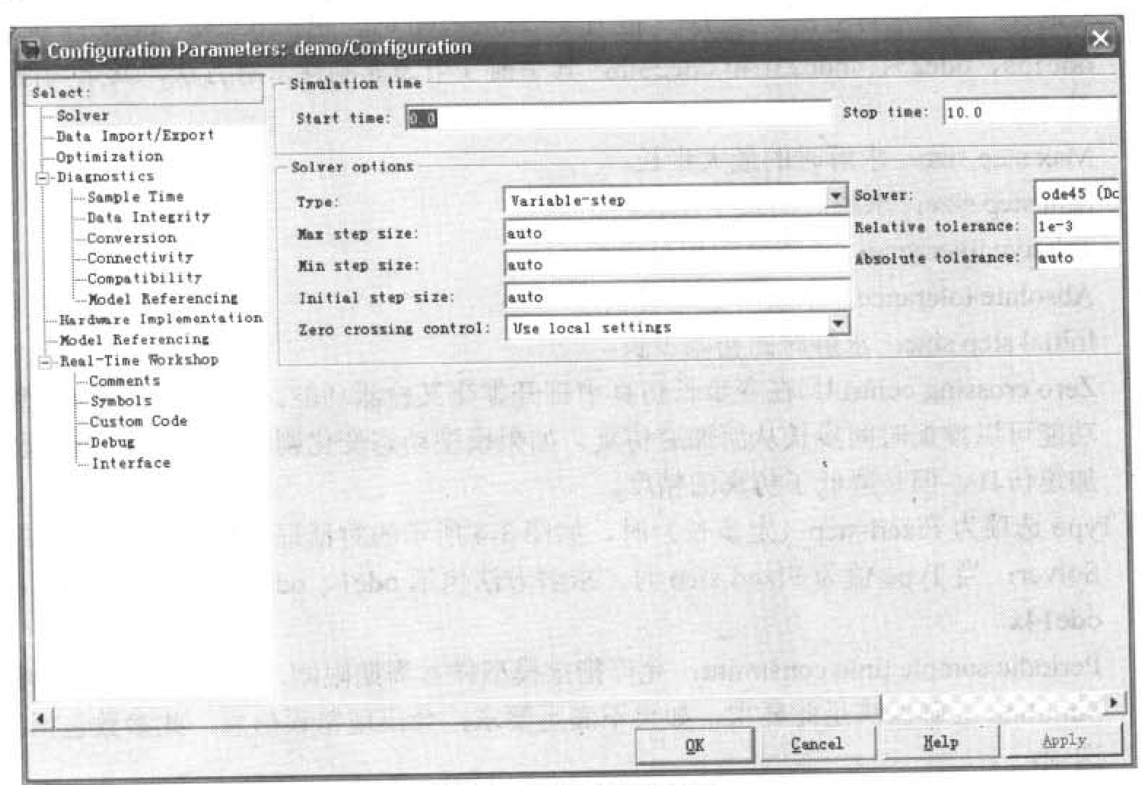


图 3-19 仿真参数设置对话框

Configuration Parameters 对话框中的各个参数包括 Solver、Data Import/Export、Optimization、Diagnostics、Hardware Implementation、Model Referencing 和 Real-Time Workshop 等。下面分别介绍各个仿真参数的配置。

3.6.1 解数器的参数设置

解数器的设置如图 3-20 所示，包括两个选项组 Simulation time 和 solver option，可以设置仿真起止时间、求解器类型、误差大小等。

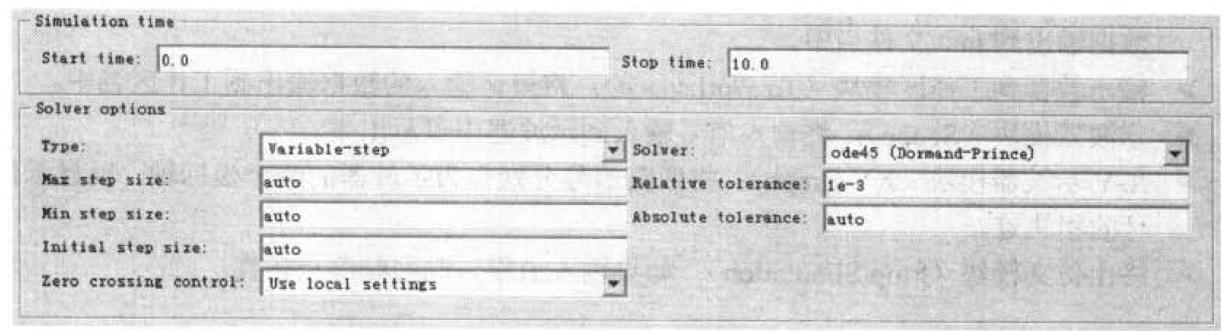


图 3-20 解数器的设置

（1）Simulink time 仿真起止时间设置

✓ Start time：仿真起始时间，默认为 0。

✓ Stop time：仿真终止时间，默认为 10。

（2）Solver options 仿真求解器设置

当 Type 选项为 Variable-step（变步长）时，对话框中参数的意义如下：

✓ Type：此选项包括 Variable-step 和 Fix-step，分别表示变步长和定步长。

✓ Solver：表示求解方法，当 Type 值为 Variable-step 时，包括 ode45、ode23、ode113、ode15s、ode23s、ode23t 和 ode23tb，其中前 3 个为非刚性求解方法，其余为刚性求解方法。

✓ Max step size：求解时的最大步长。

✓ Min step size：求解时的最小步长。

✓ Telative tolerance：求解时的相对误差。

✓ Absolute tolerance：求解时的绝对误差。

✓ Initial step size：求解时的初始步长。

✓ Zero crossing control：在变步长仿真中打开零交叉检测功能。对于大多数的模型，此功能可以增加时间步长从而加速仿真。如果模型动态变化剧烈，关闭这个选项能够加速仿真，但是降低了仿真的精度。

当 Type 选项为 Fixed-step（定步长）时，如图 3-4 所示的对话框中的参数意义如下：

✓ Solver：当 Type 值为 Fixed-step 时，求解方法包括 ode1、ode2、ode3、ode4、ode5、ode14x。

✓ Periodic sample time constraint：允许指定模型样本周期限制，在模型仿真过程中，Simulink 会确保满足此要求，如果不满足要求，会出现错误信息，此参数包括以下选项：

■ Unconstrained：无限制。

■ Ensure sample time independent：使模型从参考模型中继承样本时间，而不改变参考模型的各种性质。

■ Specified：确保模型运行在一系列划分的样本时间范围内。

✓ Tasking mode for periodic sample times：有 Auto、MultiTasking 和 SingleTasking 等三个选项，分别表示多任务与单任务模型，具体功能如下：

■ MultiTasking：当两个模块以不同速率运行时，即多任务仿真中，这种模式将会在检测到两个模块间出现不合法的样本速率时发出错误信号。不合法的样本速率会导致一

个任务的输出数据不能被另外一个任务使用。通过检测这种信号传输，MultiTasking模式可以帮助创建一个合法的现实多任务系统。

- SingleTasking：这种模式不会检测模块间的信号传输，当模型是一个单任务模型时，所有的信号传输都是同步的，不需要检测。
- Auto：这种模式会自动选择不同的运行模式，当模型是单任务模型时就用SingleTasking模式，当是多任务模式时就会自动选择MultiTasking。

其下还包括两个复选框，功能如下：

- Higher priority value indicate higher task priority：如果选中该复选框，模型的目标实时系统将会给不同的任务分配不同的优先权，高的优先权分配给高优先权值的模块。也就是说，会导致模型中低优先权值的模块与高优先权值的模块间的信号传输异步。如果不选中该复选框，目标实时系统将为低优先权值任务分配更高的优先值。详细内容可参看 Real-Time Workshop 帮助文档。
- Automatically handle data transfers between tasks：如果选中该复选框，将会在模块间插入隐含速率传输模块。

3.6.2 仿真数据输入输出设置

单击 Configuration Parameters 对话框左侧目录中的 Data Export/Import 选项，右侧如图3-21 所示。

图 3-21 仿真数据输入输出设置

（1）Load from workspace：包含若干控制选项，可以设置如何从 MATLAB 工作区调入数据。

- Input：格式为 MATLAB 表达式，确定从 MATLAB 工作区输入数据。
- Initial state：格式为 MATLAB 表达式，确定模型的初始状态。

（2）Save to workspace：可以设置如何将数据保存到 MATLAB 工作区。

- Time：设置将模型仿真中的时间导出到工作区时所使用的变量名。
- State：设置将模型仿真中的状态导出到工作区时所使用的变量名。
- Output：设置将模型仿真中的输出导出到工作区时所使用的变量名。
- Final states：设置将模型仿真结束时的状态导出到工作区时所使用的变量名。

（3）Save option：包含若干控制选项，允许设置保存到工作区或者从工作区加载数据的各种选项。

- Limit data point to last：限制导出到工作区的数据个数，如 N。在仿真结束时，MATLAB 工作区只包含最后 N 个数据。
- Decimation：如果指定为 M，Simulink 则会每隔 M 个数据输出一个。
- Format：设置保存到工作区，或者从工作区载入数据的格式，包括矩阵、结构体、带有时间的结构体。
- Signal logging name：用来保存仿真过程中信号记录的变量名。

Optimization 优化选项

单击 Configuration Parameters 对话框左侧目录中的 Optimization 项，右侧如图 3-22 所示。这个优化选项组可以选择不同的选项来提高仿真性能以及产生代码的性能。

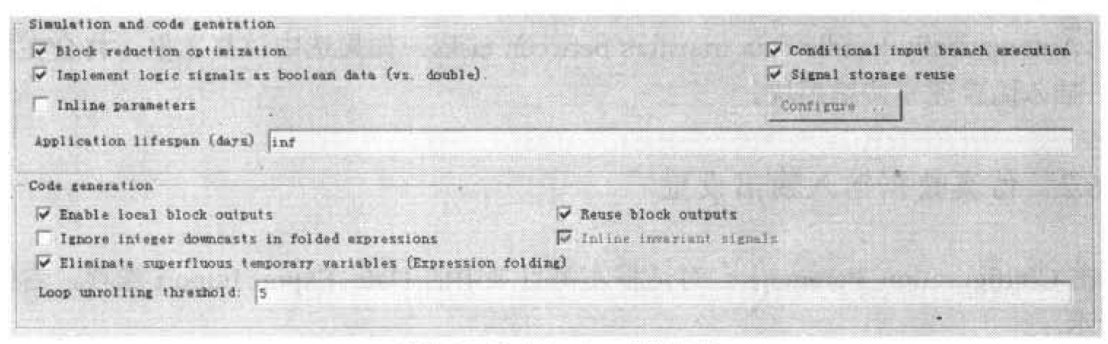


图 3-22 Optimization 优化选项

（1）Simulation and code generation：该选项组的设置对模型仿真和代码生成共同有效。

- Block reduction optimization：用一个合成模块来代替一组模块，以此来提高模型的执行效率。
- Conditional input branch execution：该选项在模型中含有 Switch 模块或者 Multiport 模块时使用。当被选中时，该选项只执行模型中那些需要计算控制输入的，以及每一个时间步长内控制输入所选择的输入数据的 Switch 或者 Multiport Switch 模块。在通过 Real-Time Workshop 生成模型代码时具有类似的功能，以提高执行速度。
- Signal Storage reuse：促使 Simulink 重新使用分配的内存来保存模块的输入与输出数据。如果不选中该复选框，Simulink 将会为每一个模块输出分配一个独立的内存，这在模型很大时将会极大地占用内存空间，因此在对模型进行调试时应该选中该复选框。如果要进行 C-MEX 函数调试，或者使用了 Floating Scope 和 Display 模块时，则不选中该复选框。
- Inline Parameters：默认在仿真过程中可以修改的可调模块参数。在仿真过程中，参数可以修改的模块称为可调模块。选中此复选框，使所有模块都称为不可调模块，可以移动这些模块到仿真循环的外部，从而加快模型仿真的速度以及模型代码的运行速度。
- Application lifespan（days）：设置模型所代表系统的活动周期。这个参数和仿真步长决定了用来保存绝对时间值的固定点模块的数据类型。

（2）Code generation：该选项组的设置只对代码生成有效。

3.6.3 仿真中异常的诊断

Diagnostics 参数配置控制面板可以配置适当的参数，如图 3-23 所示，以便在仿真执行过程中遇到异常条件时采取相应的措施。

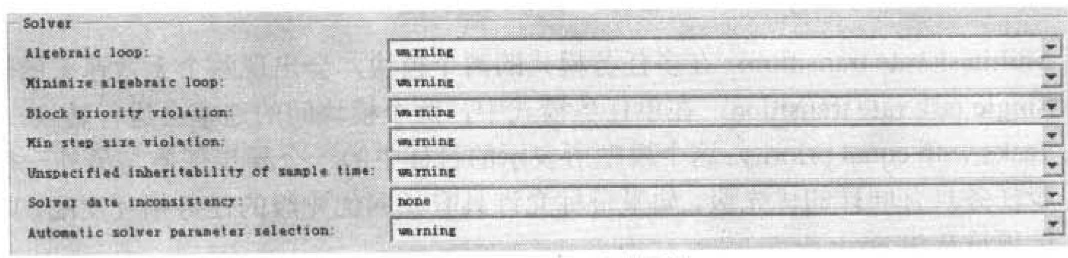


图 3-23　Diagnostics 参数配置

（1）Solver：当 Simulink 检测到与求解器相关的错误时，这个控制组可设置诊断措施。

- Algebraic loop：在执行模型仿真时可以检测到代数环。共有 3 个参数可供选择：none、warning 和 error。如果选择 error，Simulink 将会显示错误信息并高亮显示组成代数环的模块；选择 none 则不给出任何信息及提示；选择 warning 会给出相应的警告而不会中断模型的仿真。
- Minimize algebraic loop：如果需要 Simulink 消除包含有子系统的代数环及这个子系统的直通输入端口，就可以设置此选项来采取相应的诊断措施。如果代数环中存在一个直通输入端口，仅当代数环所用的其他输入端口没有直通时，Simulink 才可以消除这个代数环。
- Block priority violation：当仿真运行时，Simulink 检测优先设置错误选项的模块。
- Min step size violation：允许下一个仿真步长小于模型设置的最小时间步长。当设置模型误差需要的步长小于设置的最小步长时，此选项起作用。
- Unspecified inheritability of sample time：当模型中包含有 S 函数，但又不排除函数从父模型中继承样本时间时，指定诊断时采取的应对措施。仅当仿真过程中使用的是固定步长的离散求解器，以及求解器有周期样本时间限制时，Simulink 才会检测。
- Solver data inconsistency：兼容性检测时的一个调试工具，确保满足 Simulink 中 ODE 求解器的若干假设。其主要作用是让 S 函数和 Simulink 的内部模块具有相同的执行规则。由于兼容性检测会导致仿真性能大大降低，甚至可达到 40%，一般这个选项都设置为 none。利用兼容性检测来检测 S 函数，有利于找到出现非预期仿真结果的原因。
- Automatic solver parameter selection：当 Simulink 改变求解器参数时采取的诊断措施。假如一个连续求解器来仿真离散模型，并设置此选项为 warning，此时，Simulink 就会改变求解器的类型为离散，并在 MATLAB 命令窗口显示一个有关于此的警告信息。

（2）Sample Time：当 Simulink 检测到模型样本时间相关的编辑错误时，这个控制组可以设置诊断措施，如图 3-24 所示。

- Source block specifies-1 sample time：设置源模块的样本时间为-1，如 Sine Wave 模块。
- Discrete used as continuous：将离散模块作为连续模块，例如，单位延迟模块是一个离散模块，但是可从与其输入端相连的模块处继承连续样本属性。

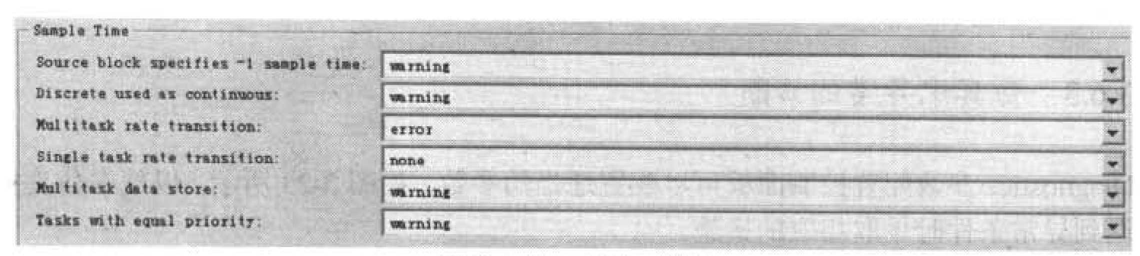


图 3-24　Sample Time 设置

- Multitask rate transition：在多任务模式的两个模块，会出现两个无效速率的转换。
- Single task rate transition：在单任务模式中，两个模块间的速率会进行转换。
- Tasks with equal priority：这个模型所表示的目标中的一个异步任务与另外一个目标异步任务具有同样的优先级。如果目标允许具有相同优先级的任务相互支配，则必须将选项设置为 error。

（3）Data Integrity：数据完整性诊断。设置当 Simulink 检测到有危害模型定义的数据完整性条件时，Simulink 所采取的诊断措施。如图 3-25 所示。

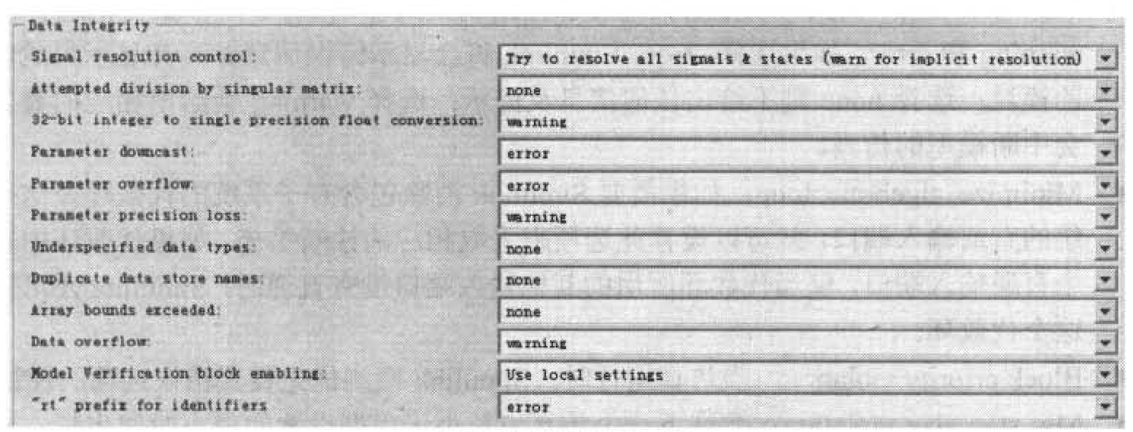


图 3-25　数据完整性诊断

- Signal resolution control：设置 Simulink 如何求解传向 MATLAB 工作区 Simulink.Signal 对象的信号。其中包括以下选项：
  - Try resolve all signals & states(warn for implicit resolution)：将求解每一个传向 Simulink.Signal 目标的信号或者状态，这些信号或者状态具有与 Simulink.signal 同样的名称。如果信号或状态隐性地求解到信号对象，就会显示警告信息。例如，一个信号对象具有与 MATLAB 工作区中已经存在的信号或状态同名，而模型又没有设置这些信号或者状态应该传输到信号对象。
  - Try resolve all signals & states：传输每一个信号或者离散状态到具有同名的 Simulink.Signal 对象中，而不管模型是否已经设置这些信号或者状态到信号对象中。
  - Use local setting：求解每一个模型指定的信号或者离散状态到 MATLAB 工作区的 Simulink.signal 中。
- Attempted division by singular matrix：product 模块通过对相乘的输入矩阵进行求逆来检测是否存在奇异矩阵。
- 32-bit integer to single precision float conversion：32 位整数转换为浮点值，这个转换会带来精度的损失。

- Parameter downcast：将模块输出的参数类型转化到具有更小值域的参数类型，例如将 uint32 转换到 uint8，这个诊断仅仅应用于可调谐的参数。
- Parameter overflow：参数溢出，参数的数据类型不能容纳参数值。
- Parameter precision loss：将模块输出转换到低精度的数据类型，例如，将 double 转换为 uint8。
- Underspecified data types：不设置数据类型。Simulink 在数据传播过程中不能判断数据的类型。
- Duplicate data types：复制数据保存所使用的变量名。
- Array bounds exceeded：这个选项将会促使 Simulink 在仿真过程中检测模块是否写到所分配内存的外部。最经典的就是当用户自己编写 S 函数存在一个漏洞时就会发生这种情况。如果是激活状态，将在模块每次执行时检测每一个模块，直接产生的影响就是降低了模型执行的速度。为了避免不必要的降低执行速度，只有当怀疑模型中编写的 S 函数存在问题时才激活这个功能。
- Data overflow：表示信号或参数值超过信号或参数数据类型所能够存储的值。
- Model Verification block enabling：这个参数可以全局，或者局部激活，或者非激活模型中的模型验证（verification）模块。其中可以选择如下选项：
  - Use local setting：根据每个认证模块的激活判定参数值来激活或者非激活模块。如果模块的激活判定值为 on，那么模块就处于激活状态，否则相反。
  - Enable all：不管模块的激活判定是如何设置的，将所有的认证模块都激活。
  - Disable all：不管模块的激活判定是如何设置的，将所有的认证模块都非激活。

（4）Conversion：该选项组用于用户设置诊断，以便在模型编译过程中 Simulink 检测到模型中存在数据转换问题时采取应对措施，如图 3-26 所示。

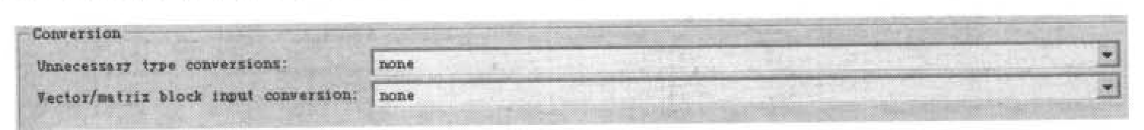


图 3-26 Conversion 设置

- Unnecessary type conversion：将 Data Type Conversion 模块添加到不需要转换数据的地方。
- Vector/Matrix block input conversion：在模块输入端口会进行向量到矩阵或者矩阵到向量的转换。

（5）Connectivity：这个选项组可以设置相应的诊断，以便在模型编译过程中 Simulink 检测到模块的连接问题时采取相应的措施，如图 3-27 所示。

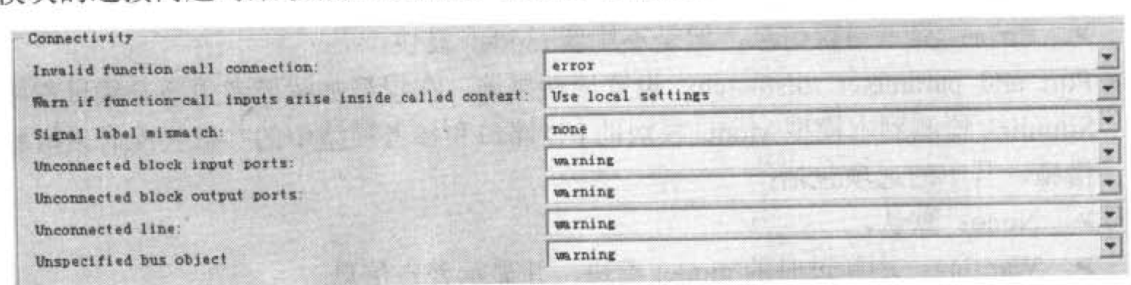


图 3-27 Connectivity 设置

- Invalid function call connection：Simulink 检测到模型中存在不正确的函数回调子系

统。如果不激活这个错误信息，可能会得到一个错误的仿真结果。

- Signal label mismatch：仿真中会遇到虚拟信号，它们具有同一个信号但是具有不同的标签。
- Unconnected block input ports：模型中包含一个模块，它有一个输入端口没有信号线与之相连。
- Unconnected block output ports：模型中包含一个模块，其中有一个输出端口没有信号线与之相连。
- Unconnected line：模型中含有一个没有连接的信号线。
- Unspecified bus object：设置相应的诊断，当 Simulink 遇到此类问题时所采取的措施，即当为参考模型生成仿真目标时，如果模型中的任何一个 Outport 模块都连接到总线，但是没有设置总线对象。

（6）Compatibility：该选项组允许用户设置相应的诊断措施，以便在模型升级或者仿真过程中检测到 Simulink 不同版本之间的不兼容性时采取相应的应对措施，如图 3-28 所示。

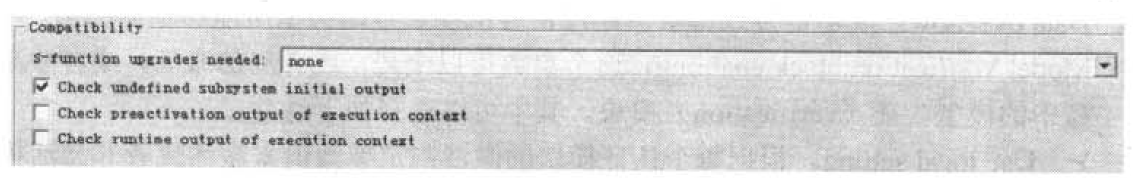


图 3-28　Compatibility 设置

（7）Model Referencing：该选项组允许用户设置相应的诊断措施，以便在模型升级或者模型仿真过程中检测到 Simulink 不同版本之间的不兼容性时采取相应的应对措施。功能类似于 Compatibility Diagnostics，只是针对的对象有所不同，如图 3-29 所示。

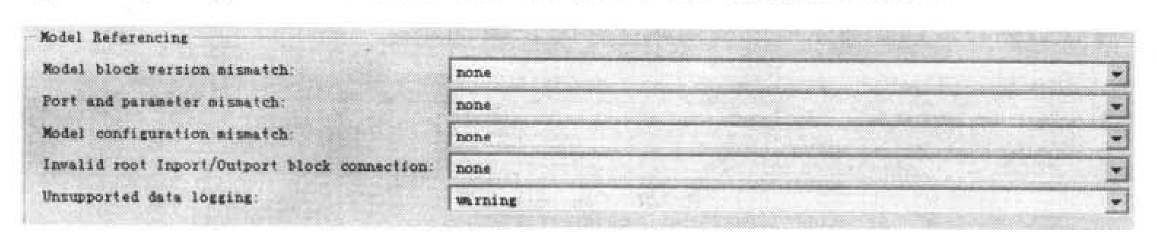


图 3-29　Model Referencing 设置

- Model block version mismatch：设置诊断措施，在模型加载或更新升级过程中，当 Simulink 检测到用来创建或更新 Model 模块的两个模型版本不兼容时所采取的对应措施。其中选项包括：
  - None：默认。
  - Warning：刷新 model 模块，同时报告警告信息。
  - Error：显示错误信息，但是不刷新 model 模块。
- Port and parameter mismatch：设置诊断措施，在模型加载或者更新升级过程中，当 Simulink 检测到本模型 Model 模块的 I/O 端口和参考模型中的参数失配时采取相应的措施，其中的选项包括：
  - None：默认。
  - Warning：刷新过时的 model 模块，并显示警告信息。
  - Error：显示错误信息，但是不刷新过时的 model 模块。
- Model configuration mismatch：设置诊断措施，在当前模型与参考模型的参数设置不

匹配，或者参考模型本身的参数设置存在问题时所应当采取的应对措施。默认设置为 none。如果怀疑模型中存在失配的参数设置并可能导致错误的结果时，可以设置诊断为 warning 或者 error。

- Invalid root import/output block connection：设置诊断措施，在代码生成过程中，当 Simulink 检测到有不合法的内部连接到模型的总线输出模块端口时所采取的应对措施。当设置为 error 时，如果遇到以下几种连接情况，Simulink 会报告错误：
  - ➢ 一个总线输出端口直接或间接地与一个以上的信号数据连接，如图 3-30 所示。
  - ➢ 一个总线输出端口与一个输入端口、Ground（地线）模块或者非数据模块连接，如图 3-31 所示。

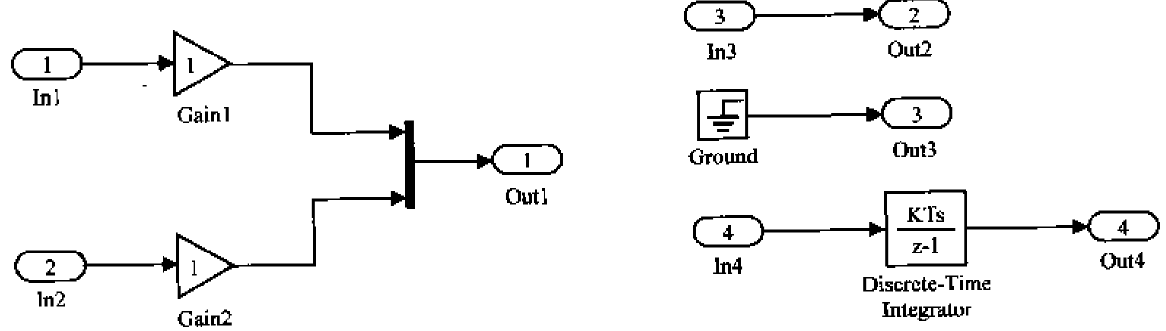


图 3-30　一个总线输出端口直接与一个以上的信号数据连接

图 3-31　一个总线输出端口与一个输入端口、Ground（地线）模块或者非数据模块连接

  - ➢ 两个总线输出模块被连接到同一个模块端口，如图 3-32 所示。
  - ➢ 一个总线输出模块不能连接到模块的某些输出部分，如图 3-33 所示。

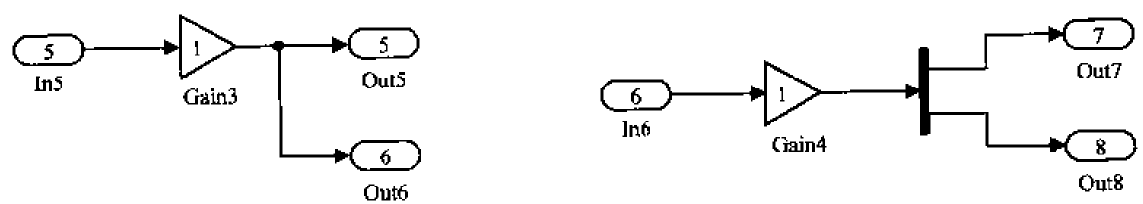


图 3-32　两个总线输出模块连接到同一个模块端口

图 3-33　一个总线输出模块不能连接到模块的某些输出部分

  - ➢ 一个输出模块不能对同一个信号数据输出两次，如图 3-34 所示。

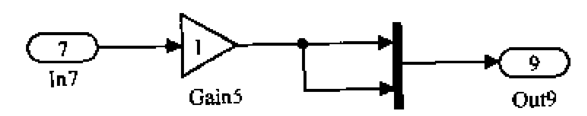


图 3-34　一个输出模块不能对同一个信号数据输出两次

- Unsupported data logging：设置诊断措施，在当前模型中包含有 To Workspace 模块或者激活输出功能的 Scope 模块时所采取的应对措施。激活输出功能的 Scope 模块指设置 Scope 输出功能。默认措施为警告用户 Simulink 不支持用此类模块从参考模型中获取数据功能。

第 4 章　运用 Simulink6.0 仿真

前面几章已经向读者介绍了 Simulink 建模仿真的大致过程，本章将向读者详细介绍运行一个模型进行仿真的相关内容。

本章主要内容：

- 确定模型的特征
- 使用菜单命令运行仿真
- 仿真参数设置
- 通过命令行运行仿真

4.1　确定模型的特征

在 Simulink 中可以建立 3 种系统模型，即连续系统、离散系统和混合系统。利用 Simulink 仿真首先要确定现实系统对应的模型，只有建立模型以后才可以确定运用 Simulink 的那部分模块建立模型。

连续系统使用微分方程描述，离散系统使用差分方程描述，离散—混合系统采用差分—微分联立方程描述。

连续系统通常都是用微分方程描述的系统，而现实世界中的多数实际系统也都是连续变化的，根据现实世界建立连续的模型，通常使用 Continuous 模块库、Math operation 模块库和 Nonlinear 模块库中的模块。

离散系统通常都是用差分方程描述的系统，而实验中，都是采用离散采样。利用 Simulink 模型建模时，通常使用 Discrete 模块库、Math operation 模块库和 Sink 模块库和 Source 模块库中的模块。

4.2　使用菜单命令运行仿真

通过菜单命令运行仿真非常简单，并且交互性比较好，这些命令使得用户不用记住命令语法就可以选择 ODE 求解器或者定义仿真参数，菜单命令的一个重要优点是当运行仿真时可以交互地做如下工作：

（1）可以改变许多仿真参数，包括停止时间、求解器和最大步长。

（2）可以改变求解器。

（3）可以同时运行其他的仿真。

（4）可以点击某条连线，在一个孤立的（没有相连）Scope 或 Display 模块中查看该条连线上传输的信号。

（5）可以改动模块的参数，只要不引起以下变化：

- 状态、输入和输出数目；
- 采样时间；
- 过零点数目；
- 任何模块参数的向量长度；
- 内部模块工作向量的长度。

在仿真期间，不能改变模型的结构，诸如增加、删除连线或模块，如果要对模型改动，必须终止仿真，修改完以后再继续运行仿真来观察结果的改变。

4.2.1 设置仿真参数和选择求解器

通过选择 Simulink 菜单下的 Parameters 菜单项，Simulink 显示 Simulation Parameters 对话框，如图 4-1 所示，用来设置仿真参数和选择求解器。Simulink 显示的仿真参数 Simulation Parameters）对话框，其中有三个页面管理如下的这些仿真参数：

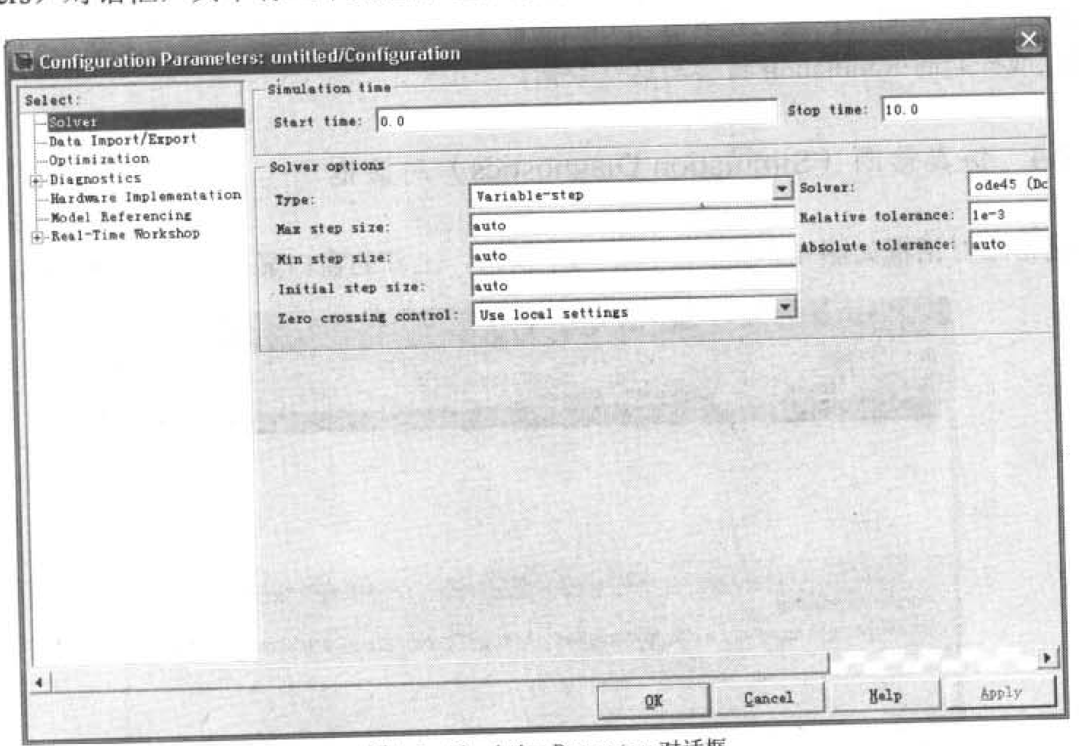


图 4-1 Simulation Parameters 对话框

（1）在 Solver 页面，可以设置开始和停止时间，选择求解器和指定求解器（Solver）的参数，另外还可以选择一些输出选项。

（2）在 WorkspaceI/O 页面，管理对 MATLAB 工作空间的输入和输出。

（3）在 Diagnostics 页面，可以选择在仿真期间显示的警告信息的层次。可以使用有效的 MATLAB 表达式设定参数，这些表达式通常由常量、工作空间变量、MATLAB 函数和数学运算符组成。

4.2.2 应用仿真参数

在设置好仿真参数并选择好求解器后，就可以在模型中使用这些设定。点击对话框底部

的 Apply 按钮，在模型中使用这些新设置的参数。要在使用参数的同时关闭对话框，点击 OK 按钮。

4.2.3 开始仿真

设定并应用仿真参数和求解器以后，就可以运行仿真了。选择 Simulation 菜单下的 Start 菜单项，或者使用键盘的快捷键 Ctrl+T 运行仿真。在选择 Start 菜单项以后，该菜单项将变成 Stop。在仿真结束的时候，计算机会发出蜂鸣信号。

Simulink 的初学者常犯的一个错误就是，在 Simulink 的模块库是活动窗口时运行仿真，在运行仿真之前必须确保模型窗口是活动窗口。要终止仿真的运行，选择 Simulation 菜单下的 Stop 菜单项，它的键盘快捷键与开始仿真的键盘快捷键相同（Ctrl+T）。

选择 Simulation 菜单下的 Pause 菜单项，可以挂起一个运行着的仿真。选择 Pause 时，该菜单项变成 Continue，可以选择 Continue 菜单项继续仿真的运行。如果模型中包含有写数据到文件或工作空间的模块，或者在 Simulation Parameters 对话框中选择了输出选项，在仿真被终止或挂起时 Simulation 将会写这些数据。

4.2.4 仿真诊断（Simulation Diagnostics）对话框

仿真诊断对话框如图 4-2 所示，包括两个窗口，上部的窗口显示如下一些信息：

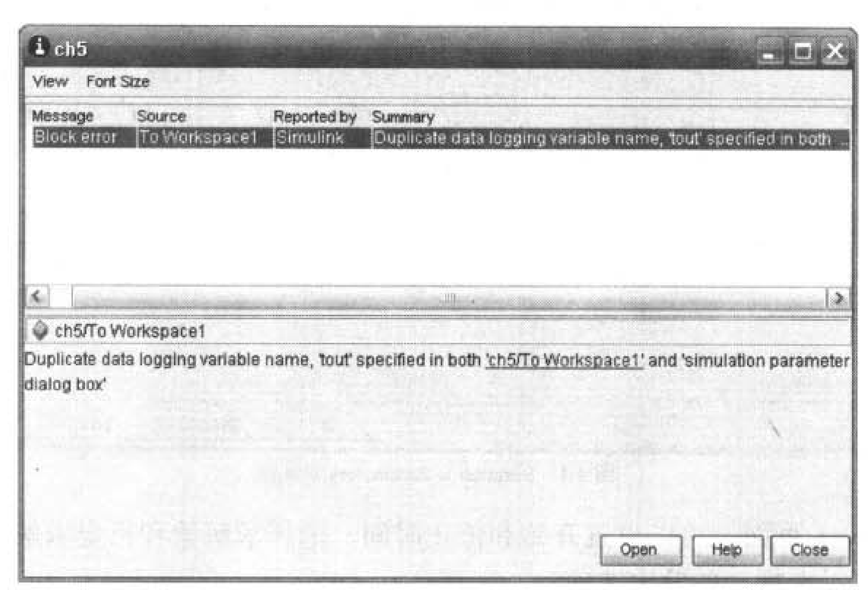


图 4-2 仿真诊断对话框

（1）Message：错误信息类型，如模块错误、警告、日志。

（2）Source：引起错误的模型元素名字，如模块名字。

（3）Fullpath：引起错误的元素路径。

（4）Reportedby：报告错误的部件，如 Simulink、Stateflow Real-TimeWorkshop 等。

（5）Summary：错误信息缩短，以适合该列显示。

仿真诊断对话框下部的窗口，在其最上部，最初包含第一个错误信息的全部内容。可以通过双击上部窗口中的其他错误信息，也可以双击错误源的名字，或按对话框中的 Open 按

钮来显示其他错误源，显示仿真诊断对话框时，必要时 Simulink 也会打开包括错误源的模型框图，并突出显示其错误源，如图 4-3 所示。

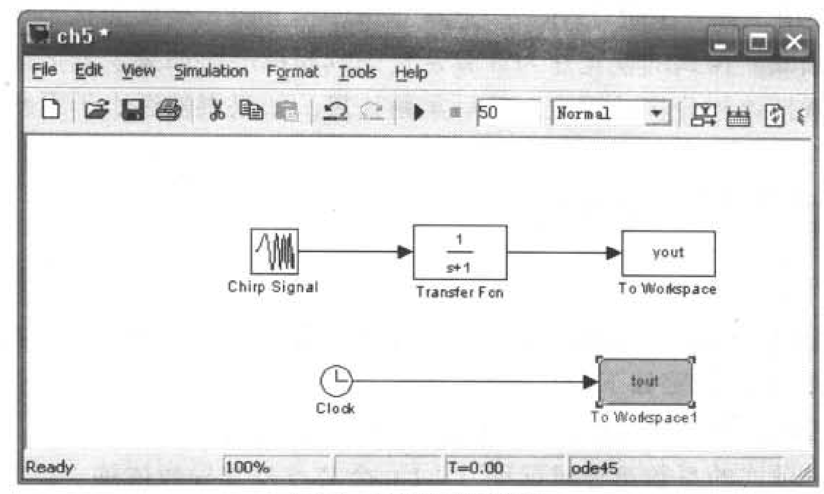


图 4-3　高亮显示错误模块

4.3　仿真参数设置

4.3.1　求解器的选择

由 Simulink 所提供的求解器都是当今国际上数值计算研究的最新成果，采用的都是速度最快、精度最高的计算方法。即便如此，也没有一个万能的计算方法，能够非常理想地求解各类微分方程。不同的系统需要利用不同的求解器，故了解系统的特性是非常重要的，如系统方程是否是刚性方程（Stiff Equation）等。下面来具体介绍各种计算方法，这对于不同的系统选择不同的方法是至关重要的。

1. ode45

这种求解器采用 Runge-Kutta 方法，这也是利用 Simulink 求解微分方程时最常用的一种方法。这种算法精度适中，是计算方法中的首选项。

它是利用有限项的 Taylor 级数取近似解函数，而误差的来源就是 Taylor 的截断项，误差就是截断误差。

ode45 分别采用 4 阶、5 阶 Taylor 级数计算每个积分步长终端的状态变量近似值，并利用这个级数的值相减，得到的误差作为计算误差的判断标准。如果误差估计值大于这个系统的设定值，那么就把该积分步长缩短，然后重新计算；如果误差小于系统的设定值，那么就将积分步长放长。

2. ode23

这种求解器采用 Runge-Kutta 方法，为了能够达到 ode45 同样的精度，ode23 的积分步长总要比 ode45 取得小。因此，ode23 处理“中度 Stiff”问题的能力优于 ode45。

ode23 是利用有限项的 Taylor 级数取近似解函数，而误差的来源就是 Taylor 的截断项，其中，误差就是指截断误差。

ode23 分别采用 2 阶、3 阶 Taylor 级数计算每个积分步长终端的状态变量近似值，并利用这个级数的值相减，得到的误差作为计算误差的判断标准。如果误差估计值大于这个系统的设定值，那么就把该积分步长缩短，然后重新计算；如果误差远小于系统的设定值，那么就将积分步长放长。

ode23 和 ode45 都是变步长算法。

3. ode113

ode113 与 ode45 和 ode23 不同，它采用的变阶 Adams 法是一种多步预报校正算法。

使用 ode113 的操作步骤如下：

（1）在预报阶段，用一个（n-1）阶多项式近似导数函数。

（2）该预报多项式的系数可通过前面（n-1）个节点及其导数值确定。

（3）用外推的方法计算下一个节点。

（4）在校正阶段，通过对前面 n 个解点和新的试探解点运用拟合技术获得校正多项式。

（5）用该校正多项式重算试探解，即获得校正解。

（6）用预报解和校正解之间的差异作为误差，与系统设定值比较，用来调整积分步长，类似于 ode45 和 ode23 方法。

ode113 在执行过程中还自动地调整近似多项式的阶数，以平衡其精确性和有效性。

ode45 和 ode23 采用的是 Taylor 级数方法，而 ode113 采用的是多项式方法，计算的次数也比前面两种方法少，所以在计算光滑系统时，ode113 速度更快。

4. ode15s

ode15s 是专门用来求解刚性（Stiff）方程的变阶多步算法，包含一种对系统动态转换进行检测的机理。这种检测使这一算法对非刚性（Stiff）系统计算效率低下，尤其是对那种有快速变化模式的系统情况。

5. ode23s

ode23s 同 ode15s 一样都是用来求解刚性方程的，是基于 Rosenbrok 公式建立起来的定阶单步算法。由于计算阶数不变，所以计算效率要比 ode15s 高一些。

6. ode23t

用来求解中度刚性方程。

7. ode23tb

用来求解刚性方程。

4.3.2 仿真性能和精度

仿真性能和精度由多种因素决定，包括模型的设计和仿真参数的选择。求解器使用它们

的缺省参数值可以使大多数模型的仿真比较精确、有效，然而，对于一些模型，如果调整求解器的仿真参数，将会产生更好的结果。如果对模型的性能比较熟悉，并且将这些信息提供给求解器，得到的仿真效果将会提高。

仿真速度慢的原因有多种。下面列举其中的一些：

（1）模型中包含有 MATLAB 的 Fcn 模块，当模型包含有 MATLAB 的 Fcn 模块时，在仿真的每一时间步都会调用 MATLAB 的解释器，这将大大地减慢仿真的速度。因此应尽可能地使用内建的 Fcn 模块或者 Elementary Math 模块。

（2）模型中包含有 M 文件形式的 S 函数。M 文件形式的 S 函数也将导致在每一时间步调用 MATLAB 的解释器，可以考虑将 S 函数转换为子系统或者 C-MEX 文件形式的 S 函数。

（3）模型中包含有 Memory 模块。使用 Memory 模块使得变阶求解器（ode15s 和 ode113）在每一时间步将阶数设为 1。

（4）最大的步长太小，如果改变了最大步长，可以试试重新使用缺省值（auto）运行仿真。

（5）对精度要求太高，默认的相对容差（0.1%）通常已经足够了。对于状态值趋于零的模型，如果绝对容差设得太小，仿真时状态值在零点附近会花去太多的时间步。

（6）时间尺度可能太长，减小时间间隔。

（7）问题可能是刚性的，而使用的是非刚性求解器，这时可用 ode15s 试一下。

（8）模型使用的采样时间相互之间不成倍数关系，相互之间不成倍数的混合采样时间会导致求解器采用足够小的步长，以保证采样时间符合所有的采样时间要求。

（9）模型包含有代数循环，在每一时间步都会反复计算代数循环，因此这会大大地降低仿真的性能。

（10）模型中将 Random Number 模块的输出传给 Integrator 模块。对于连续系统，在 Sources 库中使用 Bond-Limited Noise 模块。

要检查仿真的精度，仿真运行一段时间以后，减小相对容差到 10^{-4} 或者减小绝对容差，并重新运行它。比较两次仿真的结果，如果它们之间没有很大的差别，可以确信结果收敛。

如果经过一段时间后，仿真结果变得不稳定，可能是如下原因：

（1）系统可能不稳定。

（2）如果使用的是 ode15s，可能需要将最大的阶数限制在 2 阶（求解器稳定的最大阶数），或者试试用 ode23s 求解器。

（3）可能是相对误差或者绝对误差的设置不够理想。

如果仿真结果看起来不是很精确，可能是：

（1）对于一个拥有趋于零的状态值的模型，如果绝对容差设得太大，仿真在零状态值附近花的步数太少，减小绝对容差的大小。

（2）如果减小绝对容差不能有效地提高精度，减少相对容差的大小，减小步长，增加步数。

（3）选择适当的求解器，判断系统是否是刚性的，如果是，选择 ode15s 或者 ode23t 等求解器；如果不是，选择 ode45 等。

MATLAB 加速计算：

Simulink 和 MATLAB 是密不可分的，前面谈了如何提高 Simulink 模型的仿真速度，现在简单介绍如何提高 MATLAB 的计算速度。

MATLAB 是一种解释性语言，所以有时 MATLAB 程序的执行速度不够理想。这里将依照编者多年的实际编程经验和收集的相关资料，给出加快 MATLAB 程序执行速度的一般建议。

1. 尽量避免使用循环

循环语句及循环体经常被认为是 MATLAB 编程的瓶颈问题。改进这样的状况有两种方法：

（1）尽量用向量化的运算来代替循环操作。下面将通过例子演示如何将一般的循环结构转换成向量化的语句。

【例 4.1】 考虑无穷级数求和问题

如果要求出其中前有限项的和，如 500 000 项。

常规语句进行计算：

```
s=0;
for i=1:500000
  s=s+(1/2^i+1/3^i);
end
toc
s
```

结果：

```
s=
  1.5000
Elapsed time is 3.375000 seconds
```

如果采用向量化的方法，则可以得出结果如下：

```
s=0;
i=1:500000;
s=sum(1./2.^i+1./3.^i);
toc
s
```

结果：

```
s=
  1.5000
Elapsed time is 3.375000 seconds
```

可以看出，采用向量化的方法比常规循环运算效率要高得多，如果在一个较大的程序中有多处类似的地方，就能够明显地改善程序的执行效率。

（2）在必须使用多重循环的情况下，如果这些循环执行的次数不同，则建议将循环次数最少的放在最外一层，循环次数越多的放到靠近最内的一层，这可以提高模型的仿真速度。

【例 4.2】 考虑生成一个 10×10 000 的 Hilbert 长方形矩阵，该矩阵的定义是其第 i 行第 j 列的元素为 $h_\{I,j\}=1/(i+j-1)$。

执行代码 I，可以由下面的语句先进行 i=1:10 的循环后再进行 1:10 000 的循环。

```
for i=1:10
    for j=1:10000
```

```
        H(i,j)=1/(i+j-1);
    end
end
toc
```

结果：

Elapsed time is 3.26000 seconds.

执行代码 II：可以由下面的语句来运行。

```
for i=1:10000
    for j=1:10
        H(i,j)=1/(i+j-1);
    end
end
toc
```

结果：

Elapsed time is 22.797000 seconds.

2. 大型矩阵预先定维

给大型矩阵动态地定维是个很费时间的事情。在定义大矩阵时，首先用 MATLAB 的内在函数，如 zeros( )或 ones( )对变量进行定维，然后再进行赋值处理，这样会显著减少所需的时间。

【例 4.3】 执行代码 I：

```
H=zeros(10.10000);
for i=1:10
    for j=1:10000
        H(i,j)=1/(i+j-1);
    end
end
toc
```

结果：

Elapsed time is 0.312000 seconds.

执行代码 II：

```
J=zeros(10,10000);
for i=1:10000
    for j=1:10
        J(i,j)=1/(i+j-1);
    end
end
toc
```

结果：

Elapsed time is 0.312000 seconds.

可以看出，两个程序消耗的时间区别，和前面的分析是一致的。预先定维后，所需要的

时间显著减少了。同样一个问题，由于采用了有效的措施，执行代码 I 所需的时间就可以从 3.266s 减少到 0.312s，即效率提高了 10 倍多。执行代码 II 所需的时间就可以从 22.797s 减少到 0.359s，即效率提高了 63 倍多。这是 MATLAB7.1 执行得到的结果（Petium IV 512M 内存）。

3. 二重循环

对于二重循环这样的特殊问题，还可以使用 meshgrid( )函数构造两个 10×10 000 矩阵 i 和 j，从而直接得出 H 矩阵，更进一步地加快速度。

【例 4.4】

```
[i,j]=meshgrid(1:10,1:10000);
H=1./(i+j-1);
tod
```

结果：

```
Elapsed time is 0.031000 seconds.
```

计算时间进一步减少。

4. 优先考虑内在函数

矩阵运算应该尽量采用 MATLAB 的内在函数，因为内在的函数是由更底层的编程语言 C 语言实现的，其执行效率显然快于使用循环形式的矩阵运算。

5. 采用更加有效地算法

在实际应用中，解决同样的数学问题经常有各种各样的算法。例如求解定积分的数值算法在 MATLAB 中就提供了两个函数 quad( )和 quad8( )，其中后一个算法在精度、速度上都明显优于前一种算法。所以说，在科学计算领域是存在“多快好省”的途径的。如果一个方法不能满足要求，可以尝试其他的方法。

6. 应用 Mex 技术

虽然采用了很多措施，但执行速度仍然很慢，比如说耗时的循环是不可避免的，这样就应该考虑用其他语言，如 C 或 FORTRAN 语言。按照 Mex 技术要求的格式编写相应的部分的程序，然后通过编译链接，形成在 MATLAB 中可以直接调用的动态链接库（dll）文件，这样可以显著地加快运算速度。

7. 遵守 Performance Acceleration 的规则

关于“Performance Acceleration”的概念，可以参考 MATLAB 的帮助文件，下面将其规则总结为如下 7 条：

（1）只有使用以下数据类型，MATLAB 才会对其加速，如 logical、char、in8、uint8、int16、int32、uint32、double，而语句中如果使用了非以上数据类型则不会加速，如 numeric、cell、structure、single、fuction handle、java classes、user classes、int64、uint64。

（2）MATLAB 不会对超过三维的数组进行加速。

（3）当使用 for 循环时，只有遵守以下规则才会被加速：

- for 循环的范围只用标量值来表示；

- for 循环内部的每一条语句都要满足上面的规则，即只使用支持加速的数据类型，只使用三维以下的数组，而且循环内只调用了内建函数。

（4）当使用 if、elseif、while 和 switch 时，其条件测试语句中只使用了标量值时，将加速运行。

（5）不要在同一行中写入多条操作，这样会减慢运行速度。

（6）当某条操作改变了原来变量的数据类型或形状（大小、维数）时，将会减慢运行速度。

（7）应该这样使用复常量 x=1+2i,而不应该这样使用：x=1+2*i,后者会降低运行速度。

4.4 通过命令行运行仿真

通过命令行运行仿真与通过菜单运行仿真相比，有如下的一些优点：

（1）可以模仿 M 文件和 Mex 文件形式，甚至是 Simulink 模块图形式的模型。

（2）可以运行 M 文件来运行仿真，这样，仿真和模块参数可以反复地更改。在 MATLAB 的命令窗口或 M 文件中，输入仿真命令也可以运行仿真。在进行 Mont Carlo 分析时，可以任意地改变参数，并在一个循环中分别运行仿真，以得到不同参数的结果。可以在命令行中输入 sim 和 set_Param 命令来运行仿真。

4.4.1 使用 sim 命令

用该命令运行仿真的完整语法格式为：

[t,x,y]=sim(model,timespan,optZons?ut);

只有 model 参数是必不可少的。命令中没有提供的参数，将使用 Simulation Parameters 对话框进行的设置。

参数 option 是一个结构，它提供了附加的仿真参数，包括求解器的名字和误差容限，可以用 simset 命令定义结构参数 Options。

可以使用 set_param 命令来启动、停止、暂停或继续一个仿真，或者更新模块图。同样也可以使用 get_param 命令来检查仿真的状态,set_param 命令的格式为：

set_param('sys','simulationCommand','cmd')

式中 sys 是系统的名字，cmd 可以是 start、stop、pause、cnntinue 或 update。

get_param 命令的格式为：

getparam('sys','SimulationStatus')

Simulink 返回'stoped'，'initializing'，'paused'，'terminating'和'external'等值。

功能：仿真一个动态系统。

语法：[T,X,Y]=sim(Model,Timespan,options,Ut);

[T,K,Y1,Y2,…Yn]=sim(Model,Timespan,Options,Ut);

说明：sim 命令仿真由 Simulink 模型表达的动态系统，对描述模型的普通微分方程系统进行积分。对于模块图模型，只有 Model 参数是必需的。所有被指定为空矩阵([])的参数值都会采用 Simulaiton Parameters 对话框来设定仿真参数，而命令中指定的可选参数重置了 Simulation Parameters 对话框中设定的参数。

对于 M 文件和 Mex 文件形式的 S 函数，Model 和 Timespan 参数是必需的。对于连续状态模型，solver 参数是必须被指定的（使用 simset 命令）。对于纯离散模型，默认的 solver 参数值为 VariableStepDiscrete。

命令中参数的说明：

T　返回仿真时间向量。

X　返回仿真状态矩阵，包含连续状态和离散状态，连续状态在前，离散状态在后。

Y　返回仿真输出矩阵。对于模块图模型，每一列包含根层 Outport 模块的输出，列号对应着输出端口号，如果某一 Outport 模块的输入是一向量，那么它的输出会占几列而不是一列。

Y1，…，Yn　只有模块图模型使用这种格式。n 是根层 Outport 模块的个数，每一个模块的输出返回在相应的 Yi 中。

Model　模块图的名字。

Timespan　仿真的起始和停止时间。可以指定为下列各种形式之中的一种：

tFinal 指定停止时间，这时起始时间为 0。

[tStart,tFinal]，指定起始和停止时间。

[tStart outPutTimes tFinal]，指定起始时间和停止时间并将时间点返回给 T。通常 T 会包括更多的时间点。OutPutTimes 相当于在对话框中选择 Produce additional output。

Options 指定用 simset 命令创建的结构的可选仿真参数。

Ut 可选的模型顶层 InPort 模块的外部输入，Ut 可以是一条或多条 MATLAB 命令表达式或是一个矩阵。如果该表达式包含有时间的函数 $u(t)$，MATLAB 在每一时间步都计算它的值。如果指定的矩阵为 Ut=[T，U1，…，U2]，式中 T=[T1，…，Tn]'，第一列必须是递增的时间向量，剩下的列是相应的输入值，需要时，Simulink 在各值之间线性地插补。

4.4.2　simset 命令

功能：为 sim 命令创建或编辑仿真参数，并且设定求解器的属性。

语法：Options=simset(Property,value,…);

Options=simset(old_Opstruct,property,value,…);

Options=simset(old_Opstruct,new_opstruct);

simset

说明：simset 命令创建一个叫作 options 的结构。该结构中指定了有关的仿真参数和求解器属性的值。结构中没有指定的参数和属性值，取它们的默认值。要唯一地识别某一参数和属性，只需输入最前面的足够多的字符就可以了。输入的字符不分大小写。

Options=simset(property,value,…)：设置被指明的属性的值，并将它保存在 Option 结构中。

Options=simset(old_Opstruct,Property,value,...)：修改已经存在的结构 old_opstruct 中指明的属性的值。

Options=simset(old_Opstruct,new_Opstruct)；将已经存在的选项结构 old_Opstruct 和 new_Opstruct 合并成 options，任何 new_Opstruct 中定义的属性将重写 0ld_opstruct 中定义的相同的属性。

不带任何参数的 simset 显示所有属性的名字和它们的值。

下面列举的这些属性和参数是不能够用 get_param 和 set_param 命令来获得或设置它们的值的：

（1）AbsTol（绝对误差容限）：正标量，默认值为 10^{-6}，这一标量适用于状态向量的所有元素，AbsTol 只应用于变步长求解器。

（2）Decimation（输出变量的降采样因子）：正整数，默认值为 1，降采样因子适合于返回变量 t、x 和 y，值为 1 的降采样因子将返回所有时间点，值为 2 的降采样因子每隔一个返回一个时间点。

（3）DstWorkspace（赋变量于何处）：'base' 或 'curent' 值 'parent'。这一属性确定在 To worksPace 模块中定义为返回变量或输出变量的任何变量赋的值保存于哪一个工作空间中。

（4）FinalStateName（最终状态变量的名字）：字符串，默认为''。这一属性指定 Simulink 在仿真结束时保存模型状态的变量的名字。

（5）FixedStep（定步长）：正标量，这一属性只适用于定步长求解器。如果模型中包含有离散的组件，默认值将是基采样时间，否则将是仿真间隔的 1/50。

（6）InitialSate（连续或离散状态的初始值）：向量，默认为[ ]。初始状态包含连续状态（如果有的话）和离散状态（如果有的话），连续状态在前，离散状态在后。InitialState 取代模型中指定的初始状态，默认的空矩阵使得初始状态值取模型中指定的值。

（7）InitialStep（建议的初始步长）：正标量，默认值为 'auto'，这一属性只适用于变步长求解器，默认情况下，求解器自动地确定初始步长。

（8）MaxOrder（Odel5s 的最大阶数）：可以是 1、2、3、4 或 5，默认值为 5，这一属性只适用于 Ode15s。

（9）MaxRow（输出行数的限制）：非负整数，默认值为 0，这一属性限制，t、x 和 y 中返回的行数为最后的 MaxRows 个时间点，如果使用默认的 0 值，将没有限制。

（10）Maxstep（步长的最大值）：正标量，默认值为 'auto'，这一属性只适用于变步长求解器，并且默认值是仿真间隔的 1/50。

（11）OutputPoints（确定输出点数）；'specified' 或者 'all'，默认为 'specified'，当设为 'specified' 时，求解器只生成在 timespan 中指定的时间点的输出 t、x 和 y。当设为 all 时，t、x 和 y 还包含求解器所有采用的时间步。

（12）OutputVariablesrs（设置输出变量）：可以是 txy、tx、ty、xy、t、x 或 y，默认为 txy，如果在属性字符串中没有"t"、"x" 或 "y"，求解器将在相应的输出 t、x 或 y 中生成一个空的矩阵。

（13）Refine（输出细化因子）：正整数，缺省为 1，这一属性通过指定的因子增加输出点的数目，以产生更为光滑的输出，Refine 只适用于变步长求解器，如果指定了输出的时间，它将被忽略。

（14）RelTol（相对误差容限）：正标量，默认值为 10^{-3}，这一属性适用于状态向量的所有元素，每一积分步的估计误差满足：e(i) <= max(RelTol*abs(x(i))，AbsTol(i))，这一属性只适用于变步长求解器。

（15）Solver（时间增加方法）：可以是 VariableStepDiscrete、ode45、ode23、odell3、ode15s、ode23s、FixtedStepDiscrete、Ode5、ode4、ode3、ode2 或 odel，这一属性指定在前面的时间中使用的求解器。

（16）SrcWorkspace（在何处计算表达式）：可以是 'base'、'current' 或 'parent'，默认

为‘base’，这一属性指定在哪一个工作空间中计算模型中定义的 MATLAB 表达式。

（17）Trace（跟踪工具）：可以是‘minstep’、‘siminfo’或‘compile’，默认为空‘’。这一属性启用仿真跟踪工具（可以取一个或多个，多个时用逗号分开）。

指定‘minstep’跟踪标志时，当结果改变太快以至于变步长求解器下能确定步长和满足误差容限时，仿真将会结束。默认情况下，Simulink 会给出一条警告信息并继续运行仿真。

指定‘siminfo’标志，使得在仿真开始时提供仿真参数的一个简短的总结。

指定‘compile’标志，显示一个模块图模型的编译信息。

（18）ZeroCross（开/关过零点定位）：‘on’或‘off’，默认为‘on’。这一属性只适用于变步长求解器。如果设为‘off’，变步长求解器将不会检测过零点，即使模块本来就有过零点检测功能，求解器调整它们的步长也只是为了满足误差容限。

4.4.3 simget 命令

功能：取得选项结构的属性和参数。

语法：Struct=simget（model）

Value=simget(model，property)

说明：simget 命令获取指定的 Simulink 模型中的仿真参数和求解器的属性值。如果参数和属性被同一个变量名定义过，simget 返回变量的值，而不是它的名字；如果工作空间中不存在该变量，SimuLink 将发布一条错误消息。

Struct=simget(model)：返回指定模型的当前选项结构。选项结构是用 sim 和 simget 命令定义的。

value=simget(model，property)：从模型中提取被指明的仿真参数和求解器属性的值。

Value=simget(OptionStructure,property)；从 OptionStructure 中提取被指明的仿真参数和求解器属性的值。如果结构中没有指定值，将返回一个空的矩阵。Property 可以是一个包含你感兴趣的参数和属性列的单元数组。如果用的是单元数组，输出同样是单元数组。

要唯一地识别某一参数和属性，只需输入最前面的足够多的字符就行了。输入的字符不分大小写。

第 5 章　Simulink6.0 仿真结果分析

仿真结果分析是进行建模与仿真的一个重要环节，结果分析有助于模型的改进、完善，同时结果分析也是仿真的主要目的，Simulink 也提供了一些仿真结构分析的函数和命令，读者可以根据自己模型的特点和需要，利用各种工具箱来构造自己的仿真分析程序。本章主要介绍常用的 Simulink 仿真结果的分析方法。

本章主要内容：

- 观察输出轨迹
- 线性化
- 平衡点的确定 trim
- 线性化分析函数 linfun
- 动态系统平衡点分析

5.1　观察输出轨迹

绘制从 Simulink 输出的轨迹的方法有：将信号输入 Scope 或 XYGraph 模块；将输出写入变量并用 MATLAB 绘图命令；使用 To Workspace 模块将输出写入到工作空间并用 MATLAB 绘图命令绘出结果。

5.1.1　使用 Scope 模块

在仿真期间可以在 Scope 模块中显示输出轨迹，如图 5-1 所示，这一简单模型说明了如何使用 Scope 模块。图 5-2 和图 5-3 是两个 Scope 模块的显示结果。

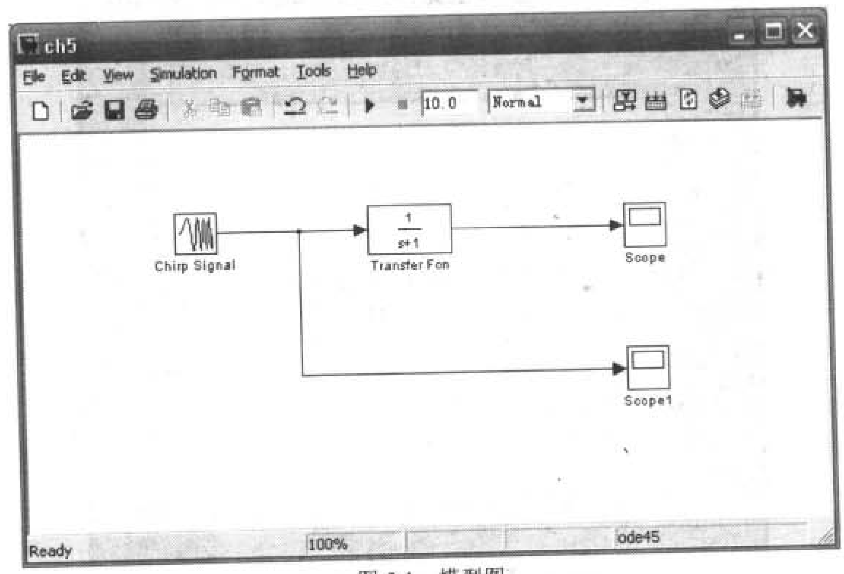


图 5-1　模型图

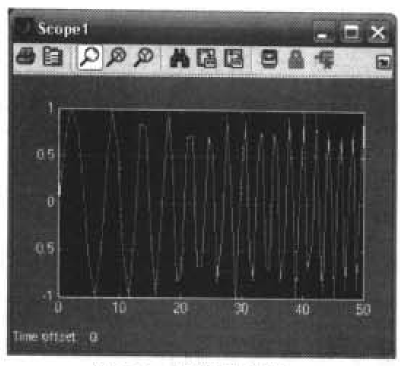


图 5-2　原信号波形

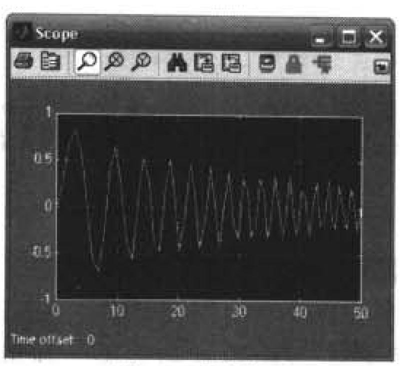


图 5-3　输出波形

Scope 窗口中显示输出的轨迹，使用 Scope 模块可以对感兴趣的部分进行放大，也可以将数据存入工作空间。

使用 XYGraph 模块可以绘出对比图，如图 5-4 和图 5-5 所示。

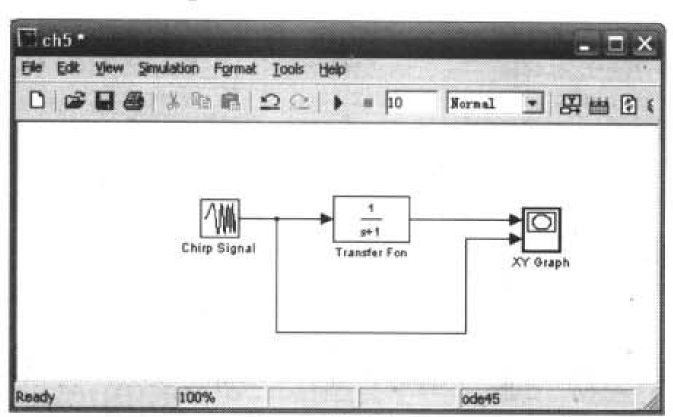


图 5-4　模型图

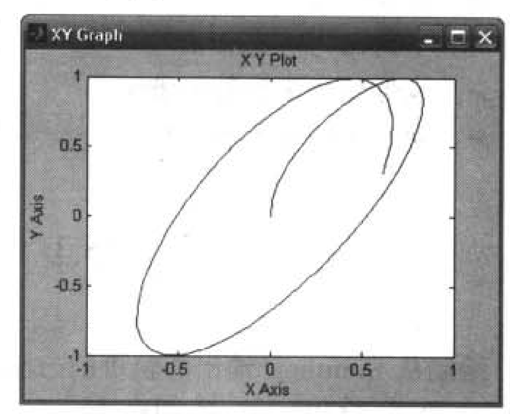


图 5-5　XYGraph 输出图形

可以通过在 Simulation Parameters 对话框的 Workspace I/O 页面中指定变量时间（tout）、输出（yout）和状态（xout）后，再通过 Simulation 菜单运行仿真。然后可以在 MATlAB 命令行中使用下面的命令，绘出结果。

plot(tout.yout)

图 5-6 是上述命令的运行结果图。

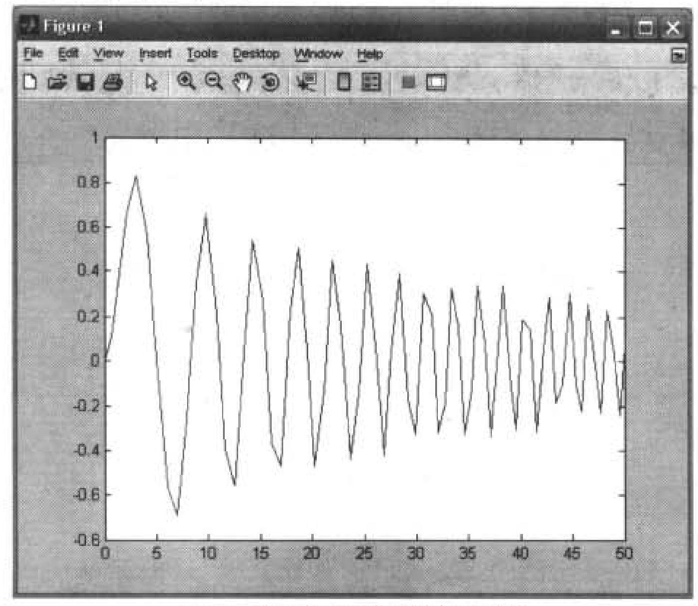


图 5-6　通过命令行得到的输出波形

5.1.2 使用返回变量

通过返回时间和输出历史数据，可以用 MATLAB 的绘图命令显示并标注输出轨迹。

图 5-7 所示模型中标为 Outl 的模块是 Signal&Systems 库中的一个输出端口模块，输出轨迹 yout 是通过积分计算器返回的。

plot(tout,yout)

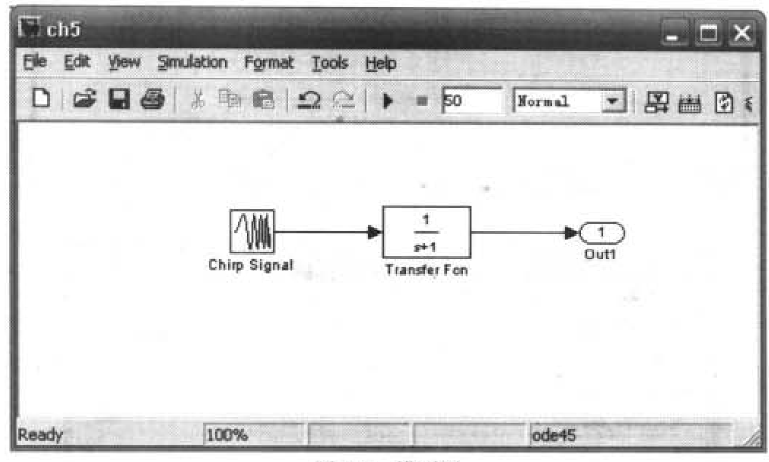


图 5-7 模型图

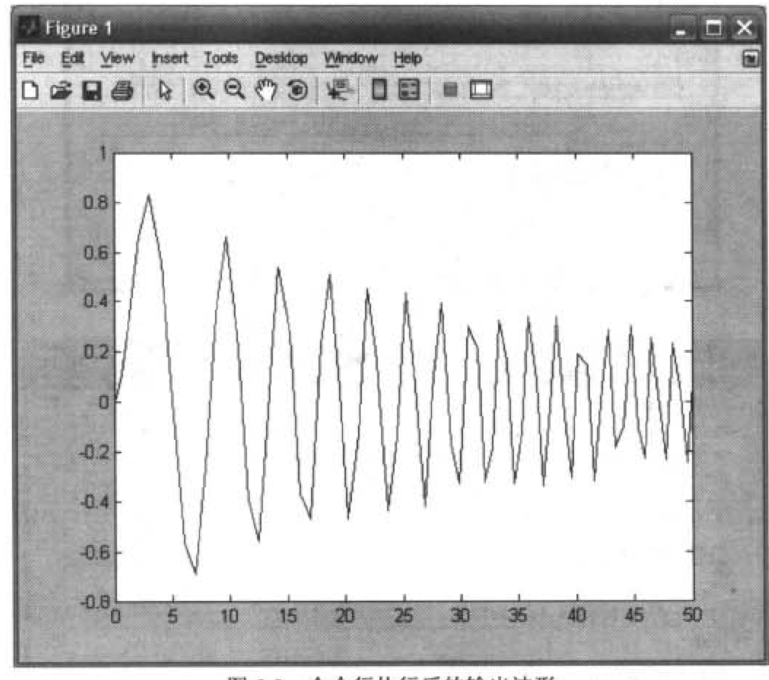


图 5-8 命令行执行后的输出波形

5.1.3 使用 To Workspace 模块

使用 To Workspace 模块可以返回输出轨迹到 MATLAB 的工作空间，如图 5-9 所示。

变量名 y 和 t 在模块参数对话框中设置（见图 5-10 和图 5-11）。当仿真结束时，将在工作空间中显示变量 y 和 t。通过将 Clock 模块输入 To Workspace 模块保存时间向量。对于菜

单驱动的仿真，通过在 Simulation Parameters 对话框的 Workspace I/O 页面中输入时间的变量名可以获得时间向量，或通过 sim 命令返回它。

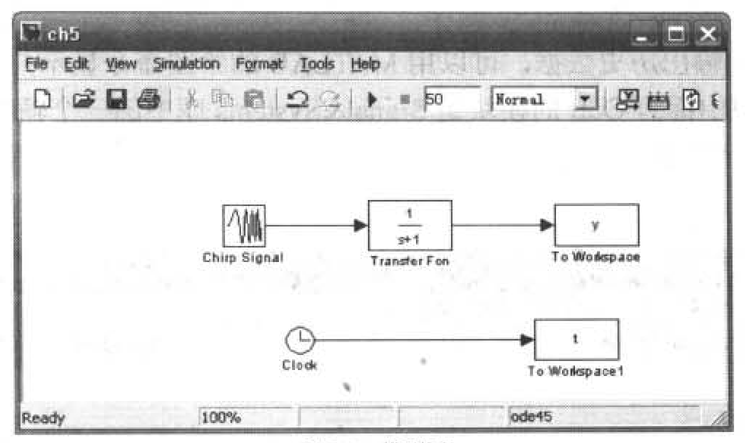


图 5-9 模型图

Sink Block Parameters: To Workspace

To Workspace

Write input to specified array or structure in MATLAB's main workspace. Data is not available until the simulation is stopped or paused.

Parameters

Variable name:

y

Limit data points to last:

inf

Decimation:

1

Sample time (-1 for inherited):

-1

Save format: Array

Log fixed-point data as a fi object

OK Cancel Help Apply

图 5-10 To Workspace 参数设置

Sink Block Parameters: To Workspace1

To Workspace

Write input to specified array or structure in MATLAB's main workspace. Data is not available until the simulation is stopped or paused.

Parameters

Variable name:

t

Limit data points to last:

inf

Decimation:

1

Sample time (-1 for inherited):

-1

Save format: Array

Log fixed-point data as a fi object

OK Cancel Help Apply

图 5-11 To Workspace1 参数设置

To Workspace 模块能够接受向量的输入，在返回的工作空间变量中，每一输入元素的轨迹存于一个列向量中。

plot(t,y)

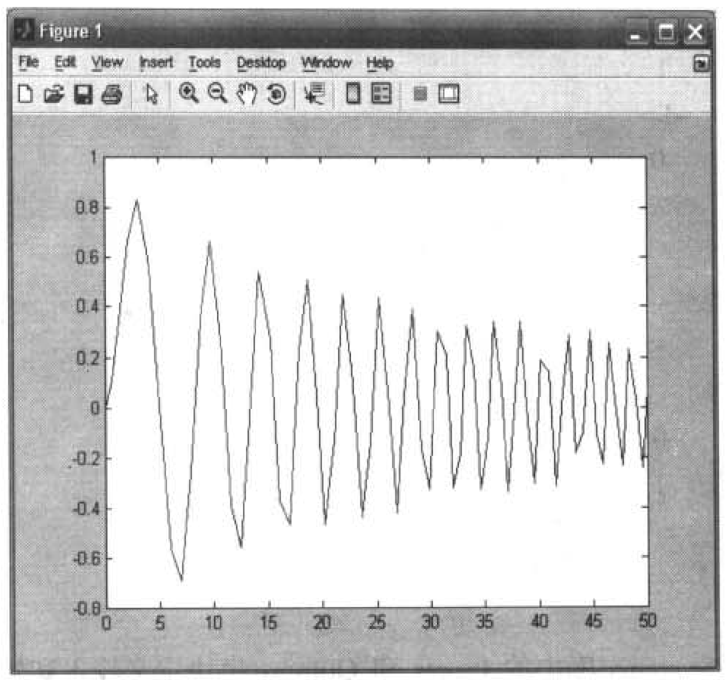


图 5-12　命令行执行结果

5.2 线　性　化

Simulink 提供 Linmod 和 dlinmod 函数以提取线性模型，模型的形式存为状态空间矩阵 A、B、C 和 D。状态空间矩阵用下面的式子描述输入与输出之间的关系：

$$x'=Ax+Bu$$

$$y=Cx+Du \tag{5.1}$$

式中 x、u 和 x'、y 分别是状态、输入和输出向量，如图 5-13 所示，模型名为 ch52。

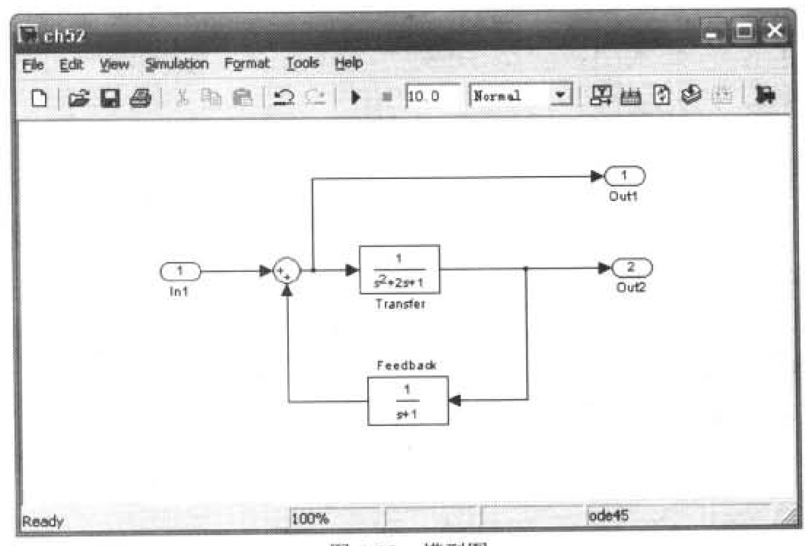


图 5-13　模型图

【例 5.1】要提取图 5-13 所示 Simulink 系统的线性模型，输入命令

```
[A B C D]=linmod('ch52')
```

输出结果：

```
A =
    -1     0     1
     1    -2    -1
     0     1     0
B =
     0
     1
     0
C =
     1     0     0
     0     0     1
D =
     1
     0
```

必须用 Signals&Systems 库中的 Inport 和 Outport 模块定义输入和输出，Source 和 Sink 模块不作为输入和输出，Inport 模块可以用来使用 sum 模块连接 Source 模块。

一旦数据成了状态空间形式或者转变成了 LTI 对象，就可以使用 Control System Toolbox 的函数进行进一步的分析。

(1) 转换成 LTI 对象

```
sys=ss(A,B,C,D)
```

【例 5.2】对于图 5.6 所示的模型，使用该命令后

```
a=
         x1   x2   x3
    x1   -1    0    1
    x2    1   -2   -1
    x3    0    1    0
b=
         u1
    x1    0
    x2    1
    x3    0
c=
         x1   x2   x3
    y1    1    0    0
    y2    0    0    1
d=
         u1
```

y1　1
y2　0

Continuous-time model.

（2）Bode 相位、幅度与频率图

bode(A，B，C，D)或者 bode(sys)

【例 5.3】 对例 5.1 或例 5.2 的结果使用 bode{sys}命令，得到图 5-14。

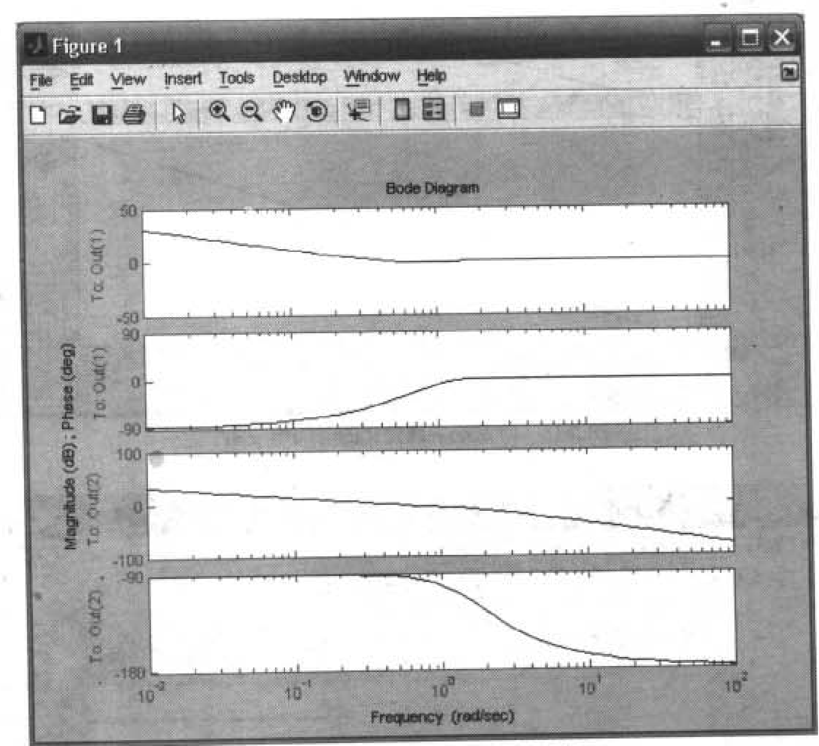


图 5-14　Bode 相位、幅度与频率图

（3）线性化的时间响应

step(A,B,C,D)或者 step(sys)

impulse(A.B,C,D)或者 impulse(sys)

lsim(A+B,C,D,u,t)或者 lsim(sys,u,t)

【例 5.4】 对例 5.1 或例 5.2 的结果，使用 step(A,B,C,D)命令得到图 5-15，使用 impulse (A,B, C,D)命令得到图 5-16。

Control System Toolbox（控制系统工具箱）和 Robust Control Toolbox（顽健控制工具箱）中其他的一些函数可以用作线性控制系统设计。

如果模型是非线性的，可能要选择一个运算点，以确定在何处提取线性模型。非线性模型对提取点的扰动大小比较敏感，这些都必须进行选择，以在截断误差和舍入误差之间取得折衷平衡。linmod 函数的其他参数是指定运算点和扰动点的。

[A,B,C,D]=linmod（'systemname',x,u,pert,xpert,upert）

对于离散系统或连续与离散混合的系统，使用 dlinmod 函数进行线性化。调用 dlinmod 的语法与调用 linmod 的语法相同，但右边第二个参数必须包含采样时间，以确定在什么时候执行线性化。

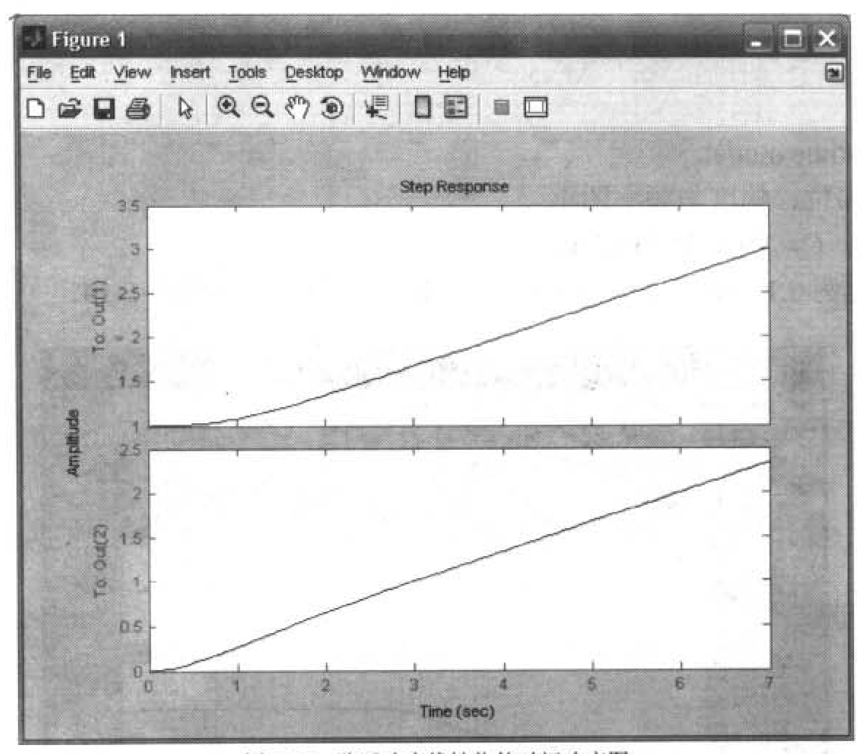


图 5-15　阶跃响应线性化的时间响应图

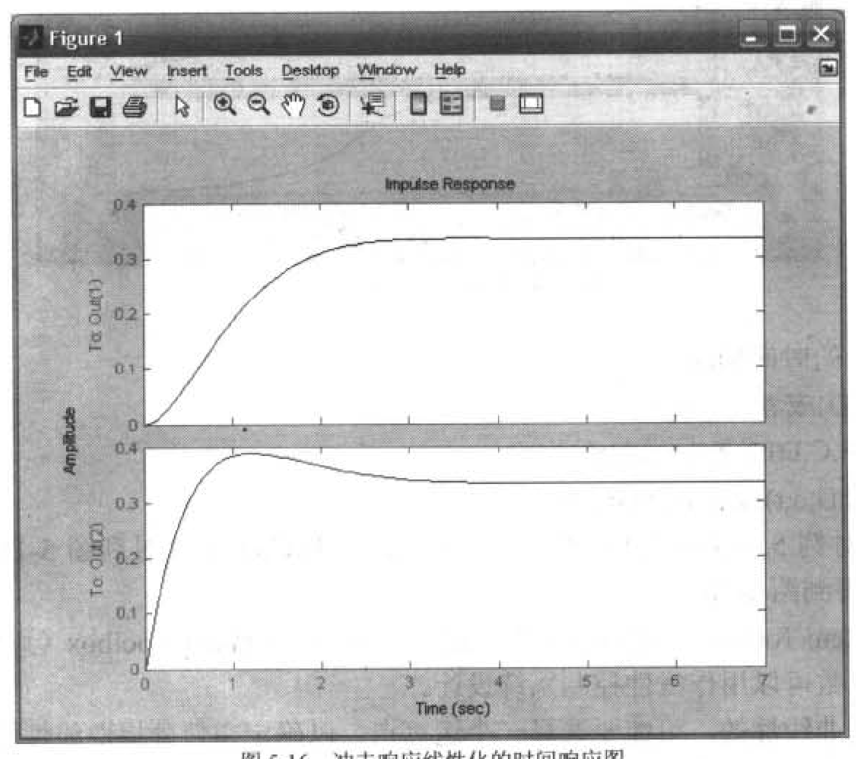


图 5-16　冲击响应线性化的时间响应图

使用 linmod 线性化一个包含有 Derivative 或者 Transport Delay 模块的模型，将会遇到麻烦。在线性化之前，要用特别设计的模块来代替这些模块，以避免出现问题，这些模块在 Simulink Extras 库的 Linearization 子库中，可以通过打开 Blocksets&ToolBoxes 图标来访问 Extras 库。

对于 Derivative 模块，使用 Switched 导数来进行线性化.

对于 Transport Delay 模块，使用 Switch 传输延迟以线性化，使用这一模块要用到控制系统工具箱。

当使用 Derivative 模块时，也可以将导数项与其他模块合并。如果有一个 Derivative 模块与一个 Transfer Fcn 模块串联，用一个如下形式的单一的 Transfer Fcn 模块实现将会更好一些：

$$\frac{s}{s+a}$$

在图 5-17 中，上面的两个模块可以用下面的一个模块代替。

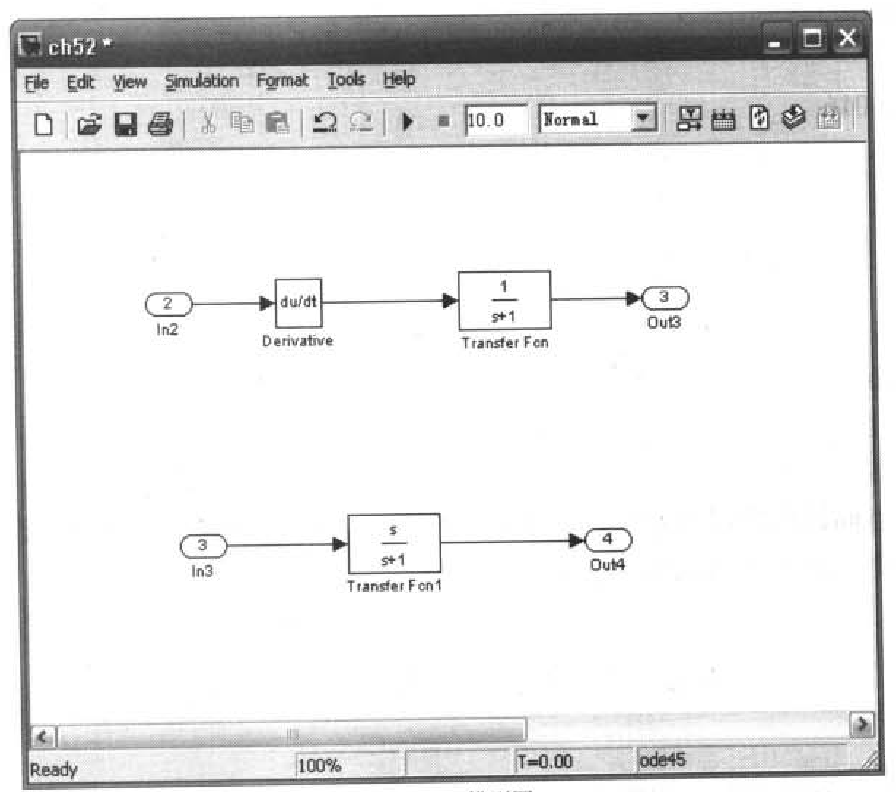


图 5-17　模型图

5.3　平衡点的确定 trim

Simulink 中的 trim 函数用来确定稳态平衡点。

【例 5.5】 考虑如图 5-13 所示的名为 ch52 的模型，使用 trim 函数找到使两个输出都为 1 的输入值和状态值。首先对状态变量 x 和输入值 u 进行初始估计，然后设定好输出 y 的值：

```
x=[0;0;0];
u=0;
y=[1;1];
```

下面的三条语句使用索引变量指明哪些变量是不变的，哪些变量是可变的：

```
ix=[]        %任何状态值可变
```

```
iu=[]        %任何输入可变
iy=[1;2]     %两个输出不能变
```

调用 trim 返回结果。由于圆整的原因，不同的计算机计算的结果可能有些不同。

```
[x,u,y,dx]=trim('ch52',x,u,y,ix,iu,iy)
```

运行以上命令后结果如下：

```
x =
     1.0000
     0.0000
     1.0000
u =
  3.0443e-016
y =
      1
      1
dx =
  1.0e-015 *
     0.2220
          0
     0.0406
```

对于平衡点问题可能没有结果，如果是这样的话，trim 首先尝试将导数设为 0，以最小化最大偏差，将它作为结果返回。

5.4　线性化分析函数 linfun

功能：在某一运算点附近提取系统的线性状态空间模型。

语法：[A,B,C,D]=linfun('systemname')

[A,B,C,D]=linfun('systemname',x,u)

[A,B,C,D]=1infun('systemnamme',x,u,pert)

[A,B,C,D]=linfun('systemname',x,u,pert,xpert,upert)

说明：

表 5-1 给出了各个参数说明。

表 5-1　　参数说明

参　数	说　明
linfun	linmod,dlinmod 或 linmod2
sys	将要提取线性化模型的 Simulink 系统的名字
x，u	状态和输入向量，如果被指定，它们设定提取线性模型的运算点
pert	x 和 u 要用到的可选的标量扰动因子，如果没有指定，缺省值为 10^{-5}
xpert，upert	可选的向量，用来分别设定每一个状态和输入的扰动程度，如果指定了该参数，将忽略参数 pert，第 i 个状态 x 扰动至 x(i)+xpert(i);第 j 个输入 u 扰动至 u(j)+upert(j)

函数 linmod 从表达为 Simulink 模型的普通微分方程的系统中提取线性模型，函数 linmod 以状态空间 A，B，C，D 的形式返回线性模型，它们描述了输入与输出的线性化关系，见式（5.1）。

在 Simulink 的模块图中，使用 Inport 和 Outport 模块来指明输入和输出。[A,B,C,D]=linmod('systemname')；获得 sys 在状态变量 x 和输入 u 为 0 时的运算点附近的线性模型。

函数 linmod 在运算点附近扰动状态值，以确定状态导数和输出（Jacobi 行列式）的变化速度，这一结果将用来计算状态矩阵，每一状态 $x(i)$扰动成：

$$x(i)+\Delta(i)$$

式中，$\Delta(i)=\delta(1+|x(i)|)$

同样，第 j 个输入扰动成：

$$u(j)+\Delta(j)$$

式中，$\Delta(j)=\delta(1+|u(j)|)$。

5.4.1 离散时间系统的线性化

函数 dlinmod 可以在任何给定的采样时间处线性化离散、多采样率和连续与离散混合的系统。调用 dlinmod 与调用 linmod 的语法相同，只不过需要插入采样时间作为第二个参数，以确定在哪些时间步执行线性化。例如，命令：

```
[Ad,Bd,Cd,Dd]=dlinmod('systemname',Ts,x,u);
```

在采样时间 Ts 和由状态向量 x 和输入向量 u 决定的运算点处产生一个离散的状态空间模型。要得到一个离散系统的连续模型的近似，将 Ts 设为 0。

对于由线性、多采样率和连续模块组成的系统，假如：

（1）Ts 是系统中所有采样时间的整数倍；

（2）Ts 不小于系统中最慢的采样时间；

（3）系统是稳定的。

函数 dlinmod 在转换后的采样时间 Ts 处产生有着相同频率和时间相应（对于常量输入）的线性模型。当这些条件不满足时，也有可能得到有效的线性模型。

计算线性化的矩阵 Ad 的特征值，可以知道系统的稳定性。如果 Ts>0，并且特征值位于单位圆内，则系统是稳定的，即：

```
all(abs(eig(Ad)))<1
```

同样，如果 Ts=0，并且特征值在左半平面内，则系统也是稳定的，即：

```
all(real(eig(Ad)))<0
```

当系统不稳定，而且采样时间不是其他采样时间的整数倍时，dlinmod 生成 Ad 和 Bd 矩阵，它们可能是复数矩阵。在这种情况下，从 Ad 矩阵的特征值仍然可以知道其稳定性。

使用 dlinmod 命令可以将系统的采样时间转换为其他的值，或者将线性离散系统转换为连续系统，反之亦然，使用 bode 函数可以了解连续或离散系统的频率响应。

5.4.2 线性化的高级形式

程序 linmod2 提供了线性化的一种高级形式，这一程序花的时间比 linmod 长，但可以产

生更为精确的结果。

调用 linmod2 的语法与调用 linmod 的语法相似，但功能不一样。例如，linmod2('systemname', x,u)同 linmod 一样生成线性模型；然而，它的每一个状态空间矩阵元素的扰动水平是被分别设定的，以将圆整和截断误差减到最小。

函数 linmod2 试图平衡圆整误差（产生于小的扰动水平，它产生的误差与有限精度数学有关）和截断误差（产生于大的扰动水平，它使得分段线性近似失效）。

[A，B，C，D]=linmod2('systemname'，x，u，pert)，变量 pert 指明了可使用的最低水平的扰动，缺省值为 10^{-8}。linmod2 有一个优点，就是它能检测突变点，并产生警告信息，例如：

Warning:discontinuity detected at A(2,3)

当出现这样的警告时，可以试看在另外的运算点处提取线性模型。

[A,B,C,D]=1inmod2('systemnamr',x,u,pert,Apert,Bpert,Cpert,Dpert)形式中，变量 Apert，Bpert，Cpert 和 Dpeert 是为每一个状态和输入组合设定扰动水平的矩阵；因此，Apert 中第 i、j 个元素是与获得的 A 矩阵中的第 i、j 个元素有关的扰动水平。用下面的命令可以返回缺省的扰动水平：

[A,B,C,D,APert,BPert,Cpert,Dpert]=linmod2（'systemname',x,u）

缺省情况下，系统时间被设为 0，对于依赖时间的系统，可以将变量 pert 设为一个包含两个元素的向量，其中第二个元素用来设定获得线性模型的 t 值。

当要被线性化的模型本来就是一个线性模型时，截断误差将不存在，因此，可以将扰动水平设为任何值，通常应该设为一个比较大的值，因为这可减小圆整误差。此时，运算点不会对获得的线性模型有影响。

从非线性模型转换为线性模型时，将保持状态的顺序。对于 Simulink 的系统，可以用下面的命令获得与每一状态有关的包含有模块名字的字符串变量：

[sizes,x0,xstring]=sys

其中 xstring 是一个字符串向量，其第 i 行是与第 i 个状态有关的模块的名字，模块图中输入和输出被依次编了号。

对于单输入多输出的系统，可以将它用 ss2tf 函数转换为传输函数，或者用 ss2zp 转换为零极点形式，还可以用 ss 函数将线性化模型转换为 LTI 对象，函数 ss 产生的状态空间形式的 LTI 对象，还可以用 tf 或 zpk 进一步转换为传输函数或零点一极点一增益的形式。

5.5 动态系统平衡点分析

功能：确定动态系统的平衡点。

语法：

[X,U,Y,DX]=TRIM('SYS')

[X,U,Y,DX]=TRIM('SYS',X0,U0)

[X,U,Y,DX]=TRIM('SYS',X0,U0,Y0,IX,IU,IY)

[X,U,Y,DX]=TRIM('SYS',X0,U0,Y0,IX,IU,IY,DX0,IDX)

[X,U,Y,DX]=TRIM('SYS',X0,U0,Y0,IX,IU,IY,DX,IDX,OPTIONS)

[X,U,Y,DX]=TRIM('SYS',X0,U0,Y0,IX,IU,IY,DX0,IDX,OPTIONS,T)

说明：从数学上讲，函数 trim 试图找到使状态导数为 0 的输入 u 和状态 x 的值。这样的点被称为平衡点，在这些点系统处于稳定状态，它经常会出现在稳定状态的动态系统中，函数 trim 从初始点开始，使用连续二次规划算法进行搜索，直到发现最接近平衡的点，必须提供初始点。如果 trim 不能找到平衡点，将返回搜索过程中遇到的状态导数的最小值，最大意义上接近于 0 的点，即返回导数最接近 0 的点。函数 trim 可以找到满足特定输入、输出和状态条件的平衡点，以及系统以特定方式改变的点，即系统状态导数等于指定的非零值的点。

由于平衡点不是惟一的，因此需要确定状态 x、输入 u 和输出 y 的特定的值。

[x,u,y] =trim('systemname')：寻找最接近系统初始状态 x0 的平衡点，找到的平衡点，是[x-x0,u,y]的绝对值的最小化点，如果 trim 不能找到接近系统初始状态的平衡点，将返回系统最接近平衡的点。特别地，它返回最小化 abs(dx-0)的点，可以用下面的命令得到 x0：

[sizes,x0,xstr]=systemname([],[],[],0)

【例 5.6】 仍以图 5-13 的 ch52 系统模型为例，输入命令：

```
[sizes,x0,xstr]=ch52([],[],[],0)

sizes =
     3
     0
     2
     1
     0
     1
     1
x0 =
     0
     0
     0
xstr=
    'ch52/Feedback'
    'ch52/Transfer'
    'ch52/Transfer'
```

[x,u,y]=trim('systemname',x0,u0,y0)：查找最接近 x0,u0,y0 的平衡点，即最小化 abs([x-x0;u-u0;y-y0])的最大值的点。

trim('systemname',x0,u0,y0,ix,iu,iy)：查找最接近 x0,u0,y0 的平衡点，并满足一系列状态、输入、输出条件。整数向量 ix,iu 和 iy 选出 x0,u0 和 y0 中需要满足的元素值。如果不能找到严格满足指定的一系列条件的平衡点，将返回满足条件最接近的点，即：

abs([x(ix)-x0(ix);u(iu)-u0(iu);y(iy)-y0(iy)])。

trim 使用一种有限制的优化方法，它限定状态导数为 0，并计算一个由 x，u 和 y 的期望值形成的最小最大值问题。对于这样的一个问题，可能不存在可行的答案，如果是这样的话，

trim 尽量使状态导数偏离 0 的值为最小。

[x,u,y,dx]=trim('systemname',x0,u0,y0,ix,iu,iy,dx0,idx)：查找指定的非平衡点，即系统状态导数具有指定的非 0 值，其中，dx0 表示在搜索开始点指定的状态导数值，idx 是在 dx0 中选择的必须严格满足的元素的索引。

函数 trim 采用一个可选的变量 options，是一个优化参数的数组，是函数 trim 用来传递到优化函数中用于查找平衡点的，优化函数依次使用这一数组来控制最优化过程，并返回过程信息。通过这种方法陈述潜在的优化过程，函数 trim 允许监视和微调平衡点的搜索。

有五个优化数组元素对平衡点搜索特别有用，表 5-2 给出了它们的值及说明。

【例 5.7】 对于一个如式（5.1）的线性状态空间模型，一个名叫 systemname 的系统如图 5-18 所示。

表 5-2　　　　　　　　　　优化数组元素

序　号	缺 省 值	说　明
1	0	指定显示选项：0 指定不显示；1 指定显示输出；-1 禁止警告信息
2	0.0001	要终止搜索，平衡点计算必须达到的精度
3	0.0001	要终止搜索，搜索目标函数必须达到的精度
4	0.0001	要终止搜索，状态导数必须达到的精度
10	N/A	返回用于平衡点搜索所使用的迭代数

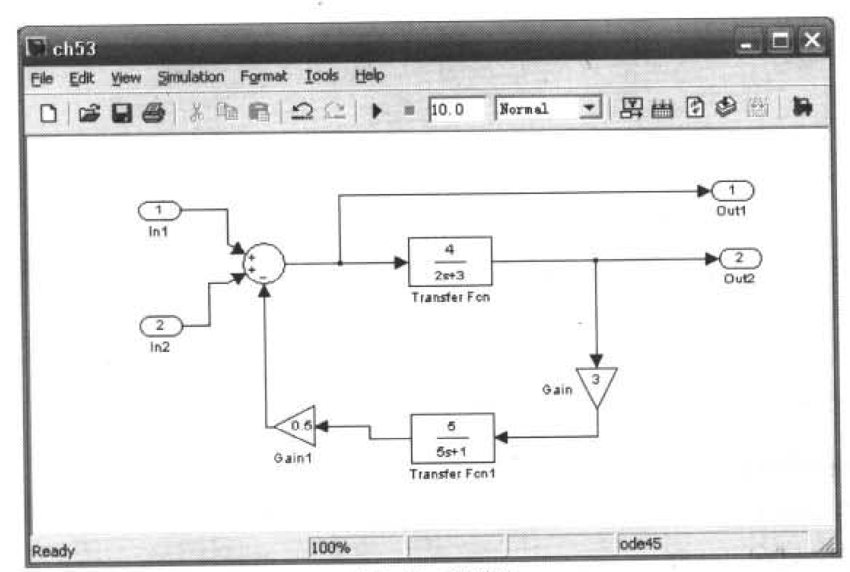


图 5-18　模型图

（1）提取线性模型

```
[A,B,C,D]=linmod('ch53')
A =
     -0.2000     6.0000
     -0.5000    -1.5000
B =
          0        0
```

```
        1       1
C =
    -0.5000          0
          0     2.0000
D =
        1       1
        0       0
```

（2）Bode 相位、幅度与频率图

```
bode(A,B,C,D);
```

得到图 5-19 所示的图形。

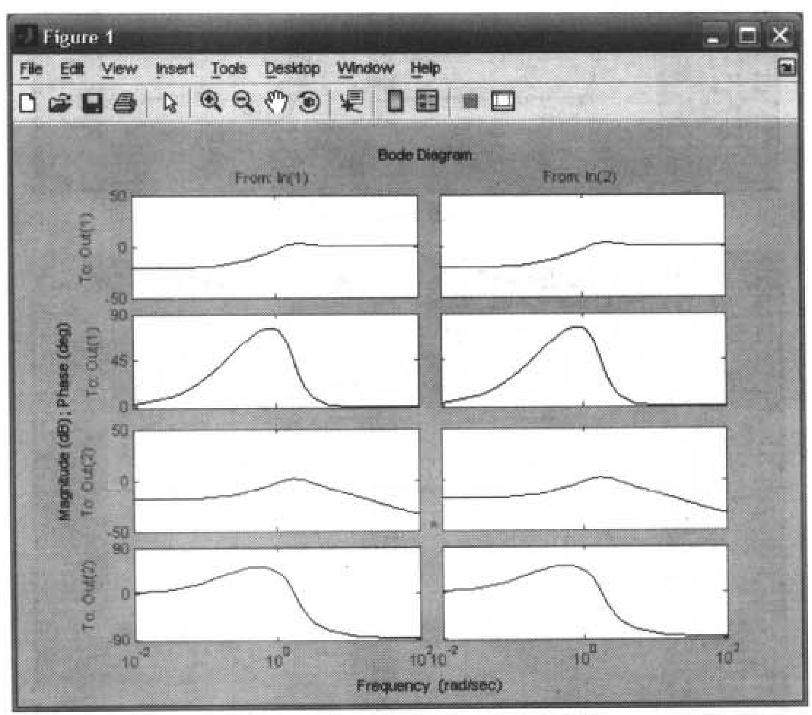


图 5-19　Bode 相位、幅度与频率图

（3）时间响应

```
step(A,B,C,D);
figure;
impulse(A,B,C,D);
```

阶跃响应和冲激响应分别如图 5-20、图 5-21 所示。

（4）用下面的命令求平衡点

```
[x,u,y,dx,options]=trim('ch53')
x =
        0
        0
u =
        0
        0
```

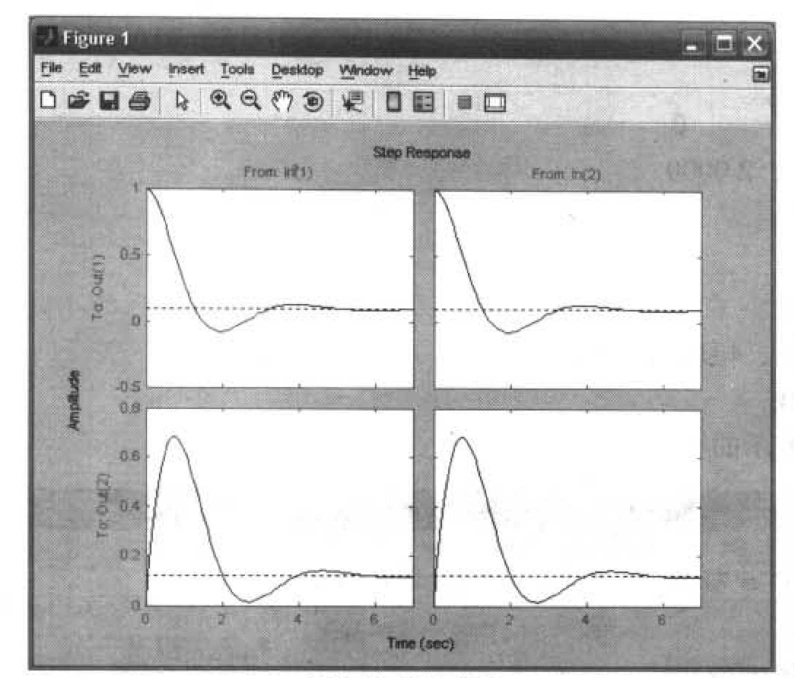


图 5-20　阶跃响应

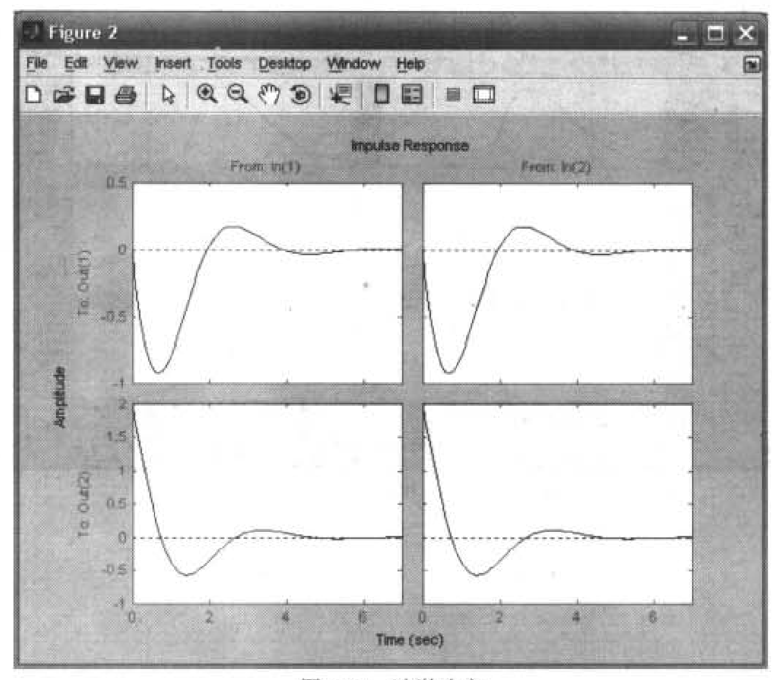


图 5-21　冲激响应

```
y =
     0
     0
dx =
     0
     0
options =
  Columns 1 through 8
         0    0.0001    0.0001    0.0000         0         0    1.0000         0
```

```
  Columns 9 through 16
         0    7.0000    2.0000         0    2.0000  500.0000         0    0.0000
  Columns 17 through 18
    0.1000    1.0000
```
迭代次数是 option(10)=7。

（5）求一个在 x=[1;1],u=[1;1]附近的平衡点

```
x0=[1;1];
u0=[1;1];
[x,u,y,dx,option]=trim('ch53',x0,u0);
x =
    1.9355
    0.0645
u =
    0.5323
    0.5323
y =
    0.0968
    0.1290
dx =
  1.0e-016 *
   -0.5551
   -0.8327
option =
  Columns 1 through 8
         0    0.0001    0.0001    0.0000         0         0    1.0000    0.9355
  Columns 9 through 16
         0   25.0000    5.0000         0    2.0000  500.0000         0    0.0000
  Columns 17 through 18
    0.1000    1.0000
```
迭代次数是 options（10）=25；

（6）求输出为 3 的平衡点

```
y=[3;3];
iy=[1;2];
 [x,u,y,dx,options]=trim('systemname',[],[],y,[],[],iy)
x =
   51.4286
    1.7143
u =
   14.1429
   14.1429
```

```
y =
    2.5714
    3.4286
dx =
  1.0e-014 *
    0.5329
    0.1776
options =
  Columns 1 through 8
         0    0.0001    0.0001    0.0000         0         0    1.0000    0.4286
  Columns 9 through 16
         0   19.0000    4.0000         0    2.0000  500.0000         0    0.0000
  Columns 17 through 18
    0.1000    1.0000
```

（7）求输出为 4，导数设为 0 和 1 的平衡点

```
y=[4;4];
iy=[1;2];
dx=[0;1];
idx=[1;2];
 [x,u,y,dx,options]=trim('ch53',[],[],y,[],[],iy,dx,idx)
x =
   60.0000
    2.0000
u =
   17.0000
   17.0000
y =
    4.0000
    4.0000
dx =
   -0.0000
    1.0000
options =
  Columns 1 through 8
         0    0.0001    0.0001    0.0000         0         0    1.0000    0.0000
  Columns 9 through 16
         0   25.0000    5.0000         0    2.0000  500.0000         0    0.0000
  Columns 17 through 18
    0.1000    1.0000
```

迭代次数是：

Options(10)=25

尽管从给定的初始点开始，找到了稳定状态的平衡点，但只是一个局部的值，有可能还有更稳定的 x，u 和 y 的值存在，因为无法保证它是一个全局最优解，除非该最优化问题是单调变化的。因此，要寻求全局的解，必须设定 x，u 和 y 的多组初始估计值分别试一下，对于有间断点的系统，trim 不能正常工作。

第 6 章　Simulink 中的系统模型

在 Simulink 中，可以建立三种系统模型，即连续系统、离散系统和混合系统。连续系统用微分方程描述，离散时间系统用差分方程描述，离散—连续混合系统采用差分—微分联立方程描述。

本章详细介绍了连续系统、离散系统和混合系统的基本模块，列举了大量实例。

本章主要内容包括：

- 连续系统模型
- 离散时间系统模型
- 离散—连续混合系统

6.1　连续系统模型

6.1.1　线性系统

连续系统通常都是用微分方程描述的系统，而现实世界中的多数实际系统也都是连续变化的。利用 Simulink 模型建模时，通常使用 Continuous 模块库、Math Operations 模块库和 Nonlinear 模块库中的模块。在此就不详细介绍每一个模块的使用方法了，具体内容参照前面章节。

由于连续系统分为线性系统和非线性系统，虽然非线性系统是绝对的，但是不利于系统的分析和设计，通常的处理方法是将非线性近似为线性系统，所以对线性系统的了解是非常必要的。

要对线性系统建模，通常都要使用到积分模块。下面通过实例介绍积分模块的使用。

【例 6.1】 直接使用积分模块缺省设置。

操作步骤如下：

（1）构造 Simulink 模型如图 6-1 所示，保存文件为 ch61.mdl，所有模块均采用默认设置。

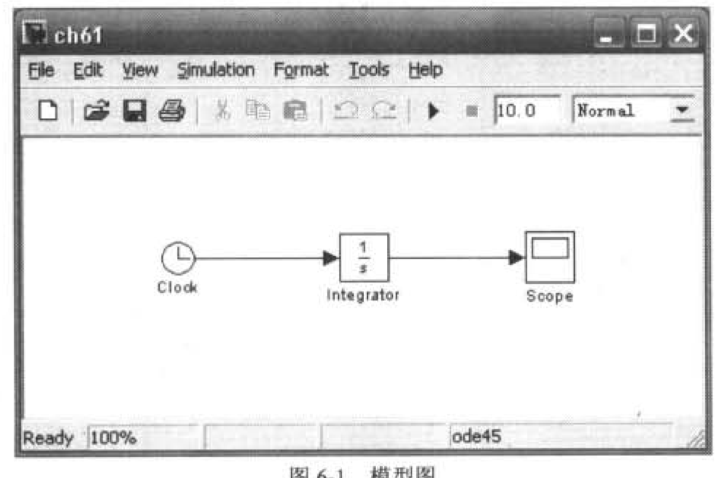


图 6-1　模型图

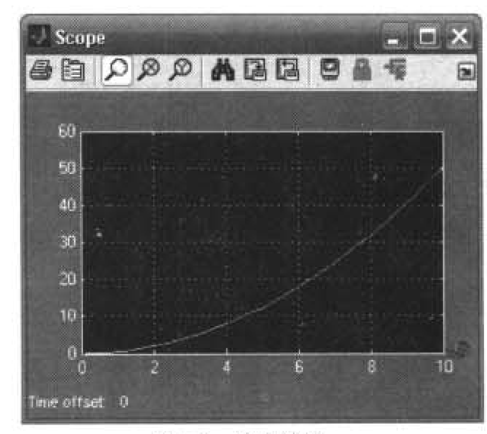


图 6-2　输出结果

（2）运行仿真，在 Simulink 窗口中选择 Simulation|start 命令，进行系统仿真。

（3）察看结果，如图 6-2 所示。

【例 6.2】 利用上升信号进行复位积分。

操作步骤如下：

（1）构造 Simulink 模型如图 6-3 所示，保存文件为 ch62.mdl。

（2）双击积分模块，在弹出的对话框中在 External reset 下拉列表框中选择 rising 选项，在 Initial condition source 下拉列表框中选择 Internal 选项，在 Initial condition 文本框中输入 5，单击 OK 按钮完成设置并关闭参数对话框。

（3）运行仿真，在 Simulink 窗口中选择 Simulation|start 命令，进行系统仿真。

（4）察看结果，如图 6-4 所示。

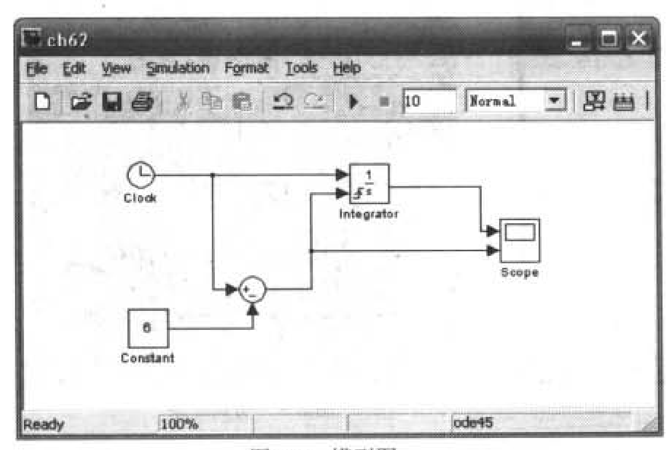


图 6-3　模型图

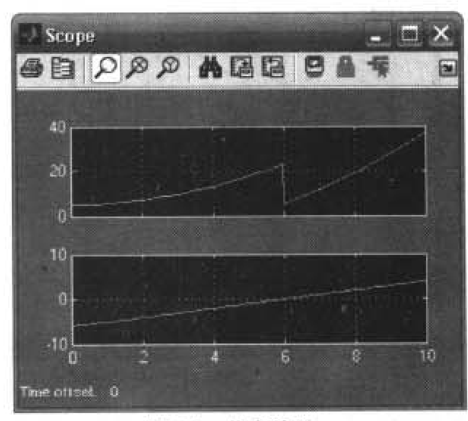


图 6-4　仿真结果

【例 6.3】 利用下降信号进行复位积分。

操作步骤如下：

（1）构造 Simulink 模型如图 6-5 所示，保存文件为 ch62.mdl。

（2）双击积分模块，在弹出的对话框中在 External reset 下拉列表框中选择 falling 选项，在 Initial condition source 下拉列表框中选择 Internal 选项，在 Initial condition 文本框中输入 5，单击 OK 按钮完成设置并关闭参数对话框。

（3）运行仿真，在 Simulink 窗口中选择 Simulation|start 命令，进行系统仿真。

（4）察看结果，如图 6-6 所示。

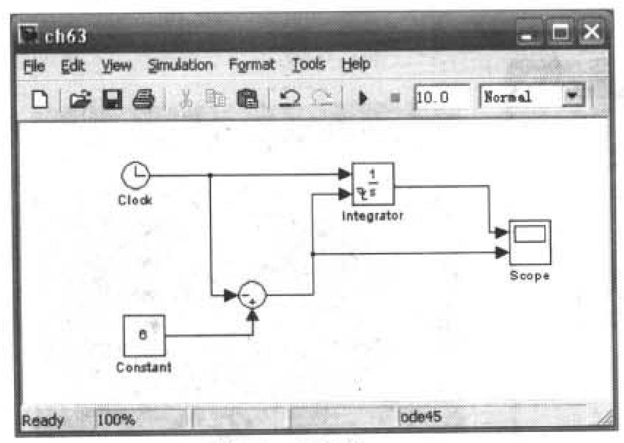


图 6-5　模型图

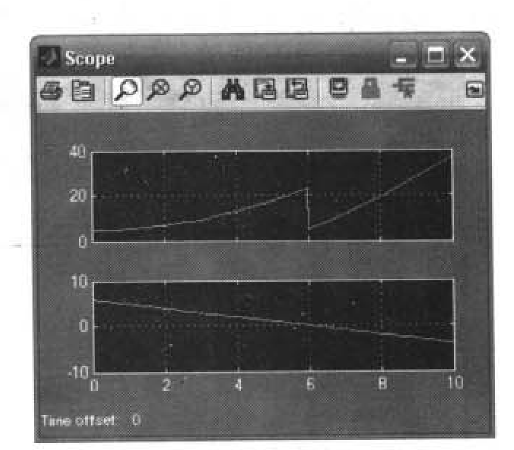


图 6-6　仿真结果

【例 6.4】 利用双向信号进行复位积分。

操作步骤如下：

（1）构造 Simulink 模型如图 6-7 所示，保存文件为 ch63.mdl。

（2）双击积分模块，在弹出的对话框中在 External reset 下拉列表框中选择 rising 选项，在 Initial condition source 下拉列表框中选择 Internal 选项，在 Initial condition 文本框中输入 5，单击 OK 按钮完成设置并关闭参数对话框。

（3）双击 Sine Wave 模块，其中 Amplitude 为 10，Frequency 为 3，单击 OK 按钮完成设置并关闭参数对话框。

（4）运行仿真，在 Simulink 窗口中选择 Simulation|start 命令，进行系统仿真。

（5）察看结果，如图 6-8 所示。

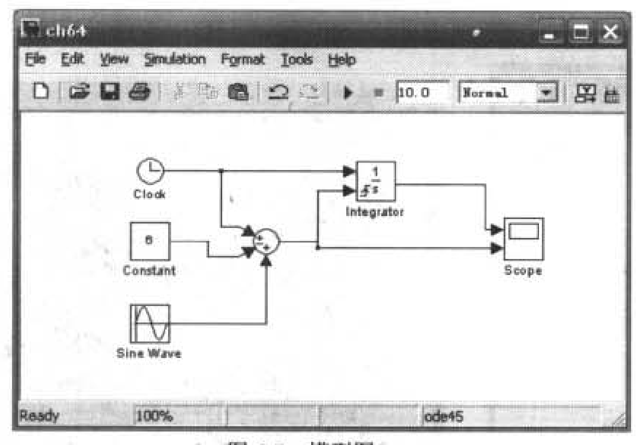


图 6-7　模型图

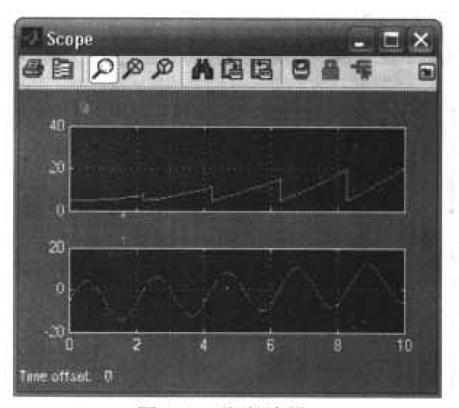


图 6-8　仿真结果

【例 6.5】利用水平信号进行复位积分。

操作步骤如下：

（1）构造 Simulink 模型如图 6-9 所示，保存文件为 ch65.mdl。

（2）双击积分模块，在弹出的对话框中在 External reset 下拉列表框中选择 level 选项，在 Initial condition source 下拉列表框中选择 Internal 选项，在 Initial condition 文本框中输入 5，单击 OK 按钮完成设置并关闭参数对话框。

（3）双击 Pulse Generator 模块，在弹出的对话框中设置 Amplitude 为 10，Period 为 3，其他采用默认设置，单击 OK 按钮完成设置并关闭参数对话框。

（4）运行仿真，在 Simulink 窗口中选择 Simulation|start 命令，进行系统仿真。

（5）察看结果，如图 6-10 所示。

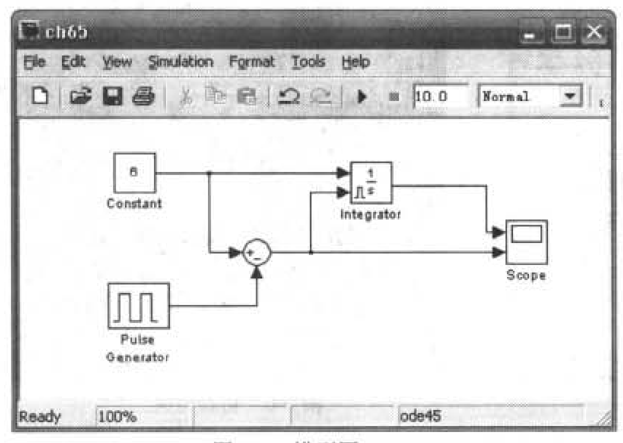


图 6-9　模型图

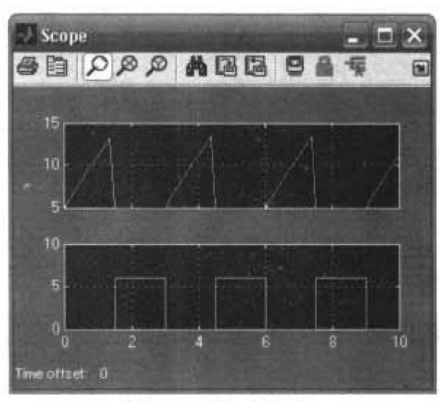


图 6-10　仿真结果

【例 6.6】利用阶跃信号进行复位积分。

操作步骤如下：

（1）构造 Simulink 模型如图 6-11 所示，保存文件为 ch66.mdl。

（2）双击积分模块，在弹出的对话框中在 External reset 下拉列表框中选择 rising 选项，在 Initial condition source 下拉列表框中选择 external 选项，在 Initial condition 文本框中输入 5，单击 OK 按钮完成设置并关闭参数对话框。

（3）添加 step 模块。

（4）运行仿真，在 Simulink 窗口中选择 Simulation| start 命令，进行系统仿真。

（5）察看结果，如图 6-12 所示。

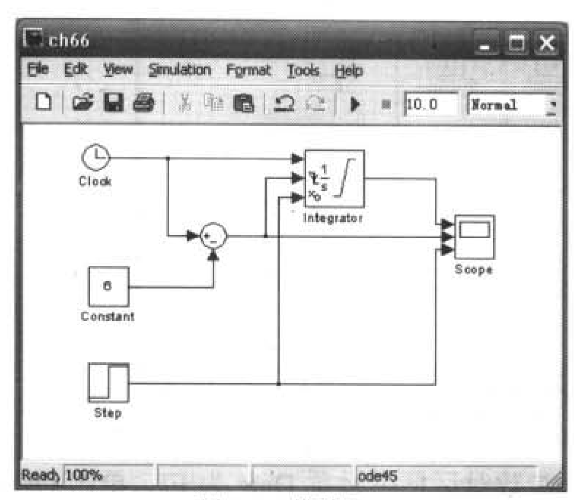


图 6-11　模型图

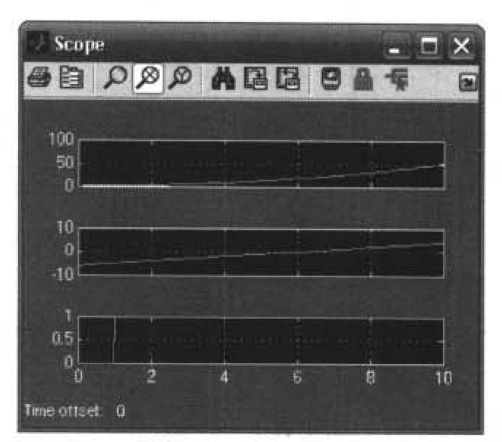


图 6-12　仿真结果

6.1.2　非线性系统

线性是相对的，非线性是绝对的。光靠 Simulink 中的线性模块是不能够完成所有任务的，所以 Simulink 还提供了大量的非线性模块，如继电器模块（Relay）死区模块（Dead zone）、饱和模块（Saturation）等。方法同线性系统类似，只要将这些非线性模块添加到模型中的适当位置即可。

6.1.3　连续系统应用实例

下面介绍三种构造连续系统模型的方法。

1. 通过积分模型构造动力学方程

实际系统可以抽象为初始状态为 0 的二阶微分方程组，下面建立一个单层楼房的受到地震影响的动力学模型。

【例 6.7】一个单层楼房，结构地震输入为美国 EI-Centro 南北方向的地震波，延续时间长 0.2s，结构质量为 M=2 923.39kg，刚度为 K=1 391.06kN/m，阻尼为 C=6 373.74Ns/m，求结构在地震作用下的地震响应。

解题步骤如下：

（1）建立楼房结构动力学方程：$M\ddot{x}(t)+C\dot{x}(t)+Kx(t)=f(t)$。

（2）变换动力学方程：$\ddot{x}=\dfrac{f(t)}{M}-\dfrac{C}{M}\dot{x}-\dfrac{K}{M}x$。

（3）构造 Simulink 模型。建立 Simulink 模型如图 6-13 所示，保存为 ch67.mdl，然后设置模块。

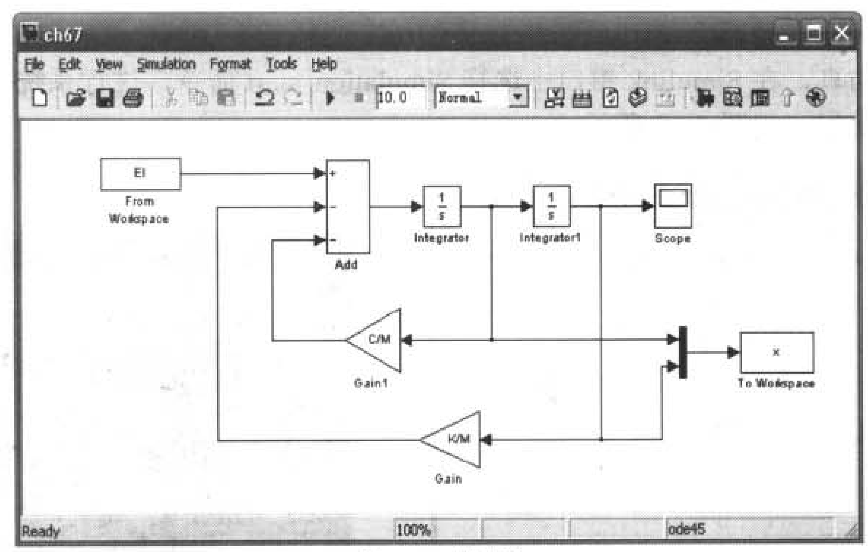


图 6-13　模型图

➢ 双击 From Workspace 模块，在弹出的参数对话框中设置 Data 为 EI，单击 OK 按钮。

➢ 双击 Gain 模块，在弹出的参数对话框中设置 Gain 为 K/M，单击 OK 按钮。

➢ 双击 Gain1 模块，在弹出的参数对话框中设置 Gain 为 C/M，单击 OK 按钮。

➢ 双击 To Worksapce 模块，在弹出的对话框中设置 Variable name 为 x，单击 OK 按钮。

➢ 积分模块保持为默认设置不变。

（4）进行仿真。编写 M 文件来输入系统仿真参数，运行 Simulink 模型。

在 MATLAB 命令窗口输入：

```
>>ch61
```

从中可以看出通过 M 文件调用 Simulink 模型的好处。对于一个参数经常修改的模型，可以通过编写一个循环来实现这个目的。

ch61.m 内容如下：

```
%ch61.m
load('EI.mat')%调入 EI 地震数据
M=2923.38;%设置系统质量
K=1391060;%设置系统刚度
C=6373.74;%设置系统阻尼
sim('ch67');%进行系统仿真

figure(1)
plot(EI(:,1),EI(:,2));%绘制地震加速度
set(gca,'FontSize',12)
```

```
xlabel('Time');
ylabel('Acceleration of Gravity')
grid on

figure(2)
plot(tout,x.signals.values(:,1));%绘制速度结果
set(gca,'FontSize',12)
xlabel('Time');
ylabel('Velocity')
grid on

figure(3)
plot(tout,x.signals.values(:,2));%绘制位移结果
set(gca,'FontSize',12)
xlabel('Time');
ylabel('Displacement')
grid on
```

（5）仿真结果。运行上述程序之后会得到如图 6-14 至图 6-16 所示的结果。图 6-14 为地震加速度曲线，也就是系统的外部输入。图 6-15 为结构响应速度曲线，图 6-16 为结构响应唯一曲线。

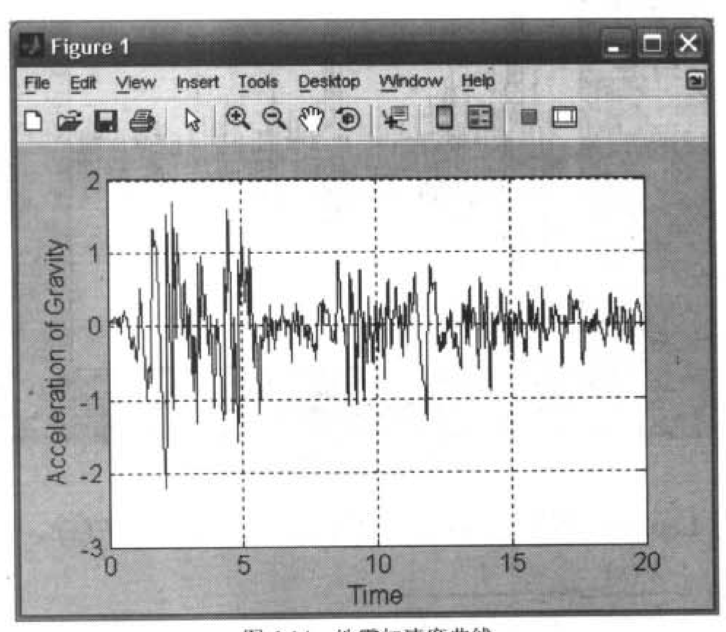


图 6-14　地震加速度曲线

2. 通过传递函数建立系统模型

【例 6.8】 依照例 6.7 中的参数，建立系统模型的传递函数。

操作步骤如下：

（1）建立动力学方程：$M\ddot{x}(t)+C\dot{x}(t)+Kx(t)=f(t)$

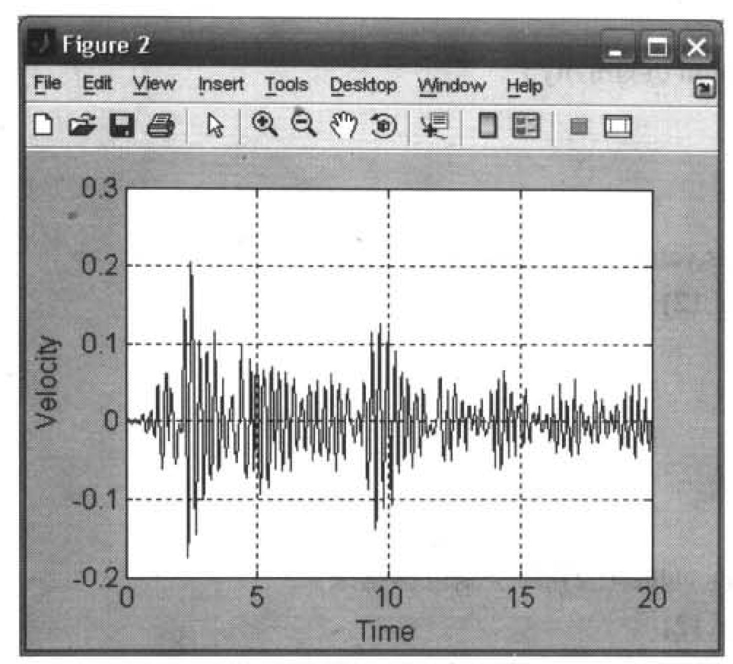


图 6-15　结构响应速度曲线

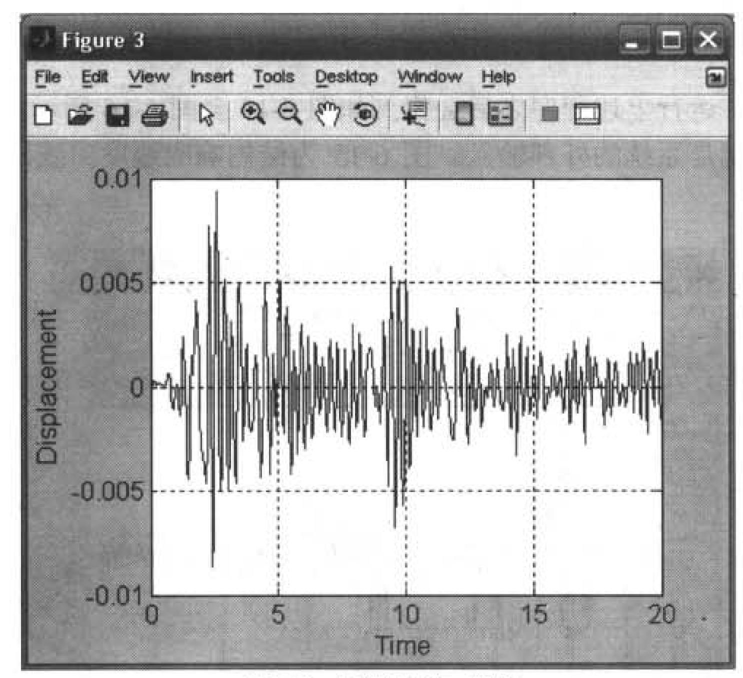


图 6-16　结构响应唯一曲线

（2）对方程进行 Laplace 变换：$s^2X(s)+CsX(s)+KX(s)=F(s)$。

整理得到：$G(s)=\dfrac{X(s)}{F(s)}=\dfrac{1}{s^2+Cs+K}$。

（3）建立 Simulink 模型。Simulink 模型如图 6-17 所示。

- 双击 From Workspace 模块，在弹出的参数对话框中设置 Data 为 EI，单击 OK 按钮。
- 双击 Transfer Fnc 模块，在弹出的参数对话框中设置 Numerator 为[1]，Denominator 为[1 C/M K/M]，如图 6-18 所示，单击 OK 按钮。
- 双击 To Worksapce 模块，在弹出的对话框中设置 Variable name 为 x，单击 OK 按钮。
- 积分模块保持为默认设置不变。

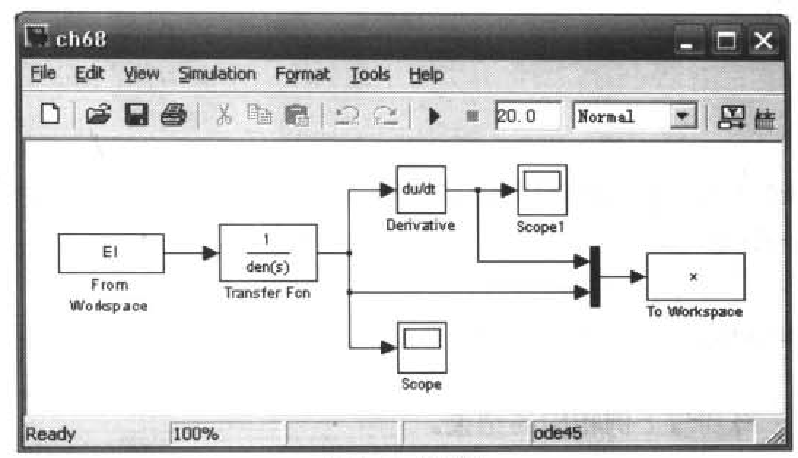


图 6-17　模型图

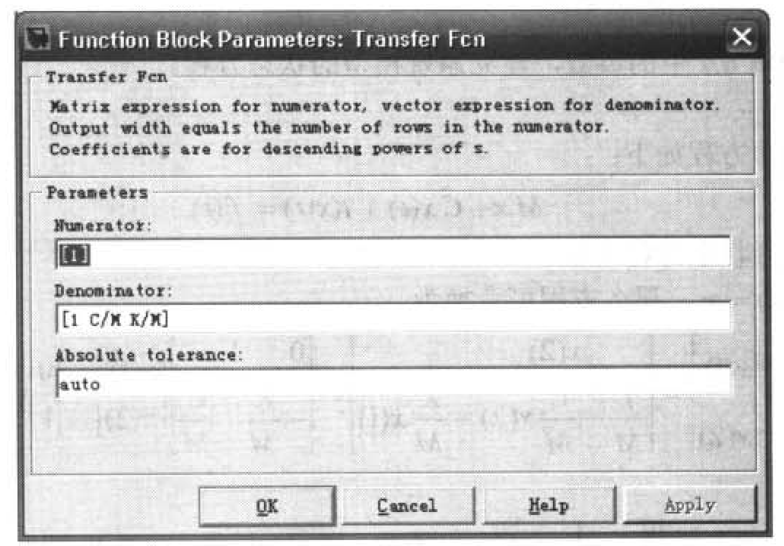


图 6-18　传输函数参数设置

（4）进行仿真。编写 M 文件来输入系统仿真参数，运行 Simulink 模型，保存文件名为 ch62.m。

在 MATLAB 命令窗口输入：

```
>>ch62
```

ch62.m 文件如下：

```
% ch62.m
load('EI.mat')%调入 EI 地震数据
M=2923.38;%设置系统质量
K=1391060;%设置系统刚度
C=6373.74;%设置系统阻尼
sim('ch68');%进行系统仿真

figure(1)
plot(tout,x.signals.values(:,1));%绘制速度结果
set(gca,'FontSize',12)
xlabel('Time');
```

```
ylabel('Velocity')
grid on

figure(2)
plot(tout,x.signals.values(:,2));%绘制位移结果
set(gca,'FontSize',12)
xlabel('Time');
ylabel('Displacement')
grid on
```

（5）仿真结果，得到与上例相同的结果。

3. 通过状态方程建立系统模型

【例 6.9】依照例 6.7 中的参数，建立系统模型的状态方程。

操作步骤如下：

（1）建立动力学方程如下：

$$M\ddot{x}+C\dot{x}(t)+Kx(t)=f(t)$$

（2）对方程进行状态方程变换。

令 $x(1)=x$，$x(x)=\dot{x}$，那么方程可变换为：

$$\dot{x}=\begin{bmatrix}\dot{x}(1)\\ \ddot{x}(2)\end{bmatrix}=\begin{bmatrix}x(2)\\ \frac{f}{M}-\frac{C}{M}x(2)-\frac{K}{M}x(1)\end{bmatrix}=\begin{bmatrix}0 & 1\\ -\frac{K}{M} & -\frac{C}{M}\end{bmatrix}\begin{bmatrix}x(1)\\ x(2)\end{bmatrix}+\begin{bmatrix}0\\ 1\end{bmatrix}\frac{f}{M}$$

整理得：

$$\begin{cases}\dot{x}=Ax+Bu\\ y=Cx+Du\end{cases}\text{其中 }A=\begin{bmatrix}0 & 1\\ -\frac{K}{M} & -\frac{C}{M}\end{bmatrix},\ B=\begin{bmatrix}0\\ 1\end{bmatrix},\ C=\begin{bmatrix}1\,0\\ 0\,1\end{bmatrix},\ D=\begin{bmatrix}0\,0\\ 0\,0\end{bmatrix}。$$

（3）建立 Simulink 模型。

Simulink 模型如图 6-19 所示，保存文件名为 ch69.mdl。

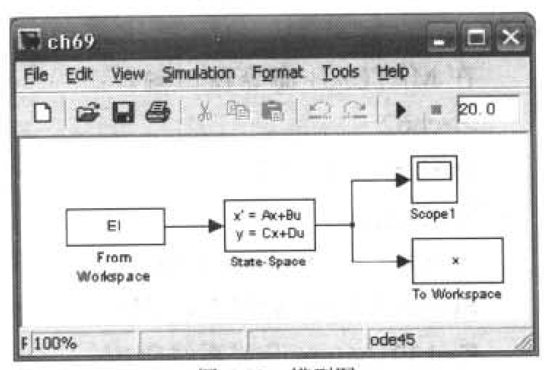


图 6-19 模型图

➢ 双击 From Workspace 模块，在弹出的参数对话框中设置 Data 为 EI，单击 OK 按钮。

➢ 双击 State-Space 模块，在弹出的参数对话框中，设置 A 为 A，设置 B 为 B，设置 C 为 C，设置 D 为 D，如图 6-20 所示，单击 OK 按钮。

➢ 双击 To Worksapce 模块，在弹出的对话框中设置 Variable name 为 x，单击 OK 按钮。

➢ 积分模块保持为默认设置不变。

（4）进行仿真。编写 M 文件来输入系统仿真参数，运行 Simulink 模型以及查看结果，保存文件名为 ch63.m。

在 MATLAB 命令窗口输入：

```
>>ch63
```

ch63.m 文件如下

```
% ch63.m
load('EI.mat')%调入 EI 地震数据
M=2923.38;%设置系统质量
K=1391060;%设置系统刚度
C=6373.74;%设置系统阻尼
A=[0 1;-K/M -C/M];
B=[0;1];
C=eye(2);
D=zeros(2,1);
sim('ch69');%进行系统仿真

figure(1)
plot(tout,x.signals.values(:,1));%绘制速度结果
set(gca,'FontSize',12)
xlabel('Time');
ylabel('Velocity')
grid on

figure(2)
plot(tout,x.signals.values(:,2));%绘制位移结果
set(gca,'FontSize',12)
xlabel('Time');
ylabel('Displacement')
grid on
```

（5）仿真结果，得到与上例相同的结果。

6.2 离散时间系统模型

离散系统通常都是用差分方程来描述的系统，而实验中，都是采用离散采样。利用 Simulink 模型建模时，通常使用 Discret 模块库、Math operation 模块库和 Sink 模块库和 Source 模块库中的模块。在此不详细介绍每一个模块的使用方法，具体可以参看前面的章节。

6.2.1 一些基本模块

在此只介绍几个主要的模块：

➢ 单位延迟模块（Unit delay）

实现计算 $y(k)=u(k-1)$，是传递函数的特殊形式，即为 $\frac{1}{z}$。

➢ 离散传递函数

传递函数主要有 3 种形式，分别如下：

- 离散传递函数模块（Discrete Transfer Fcn），函数的分子和分母都是以 z 的降幂形式升序排列的。
- 零极点传递函数模块（Discret Pole-Zero），函数的分子和分母都是以 z 因式分解的形式表示的。
- 离散滤波器模块（Discrete Filter），函数的分子和分母都是以 z^{-1} 的降幂形式升序排列的。

➢ 离散状态方程模块（Discrete State-space）

离散模块的具体形式如下：

$$\begin{cases} y(n) = Cx(n) + Du(n) \\ x(n+1) = Ax(n) + Bu(n) \end{cases}$$

确定了 A，B，C，D，就确定了离散系统，矩阵还可以通过连续系统转换 c2d 函数得到。

➢ 零阶保持器（Zero_Order hold）

输入端是采样器，输出端是常数保持器，实现 $y(kT)=u(t)$。

6.2.2 多速率离散时间系统

在离散系统中，往往会遇到同一个系统具有多个不同采样速率、不同时间偏移的子系统。例如计算机系统，其中的 CPU、串/并联控制器、磁盘驱动器、输入键盘就采用不同的工作速率。

从原理上讲，多速率系统建模与单速率系统建模没什么不同。在 Simulink 中，可以通过不同的颜色来区分不同的采样速率，这有利于用户区别对待。这个功能可以通过菜单项进行设置，选择 Format|Sample time color 命令，然后通过选择 Edit | Update Diagram 命令来更新窗口。

【例 6.10】 在实际的控制系统中，控制器和系统本身的工作频率是不一样的，控制器的频率往往会低于系统本身的工作频率。

假设某系统的离散系统方程为：

$$\begin{cases} x_1(k+1) = -0.5x_1(k) + 0.4\sin x_2(k) - u_1(k) \\ x_2(k+1) = 0.5x_1(k) - 0.035x_2(k) - u_2(k) \end{cases}$$

式中，$u(k)$是系统输入。该过程采样周期为 0.05 秒，控制器采用周期为 0.1 秒的比率控制器，显示系统的更新周期为 0.2 秒。

（1）建立 Simulink 模型。建立的模型如图 6-20 所示，保存文件名为 ch610.mdl。

- 模块 Zero-Order Hold 和模块 Zero-Order hold1 采样为 0.2。
- 模块 Zero-Order Hold2 和模块 Zero-Order hold3 采样为 0.1。
- 模块 Add2、模块 Add3、模块 Gain2 和模块 Gain3 的 Sample time（采样时间）为 0.1，其他保持默认设置。
- Sine Wave1 参数对话框中设置 Amplitude 为 2，Sample time 为 0.1。
- Sine Wave 参数对话框中设置 Amplitude 为 1，Sample time 为 0.1。

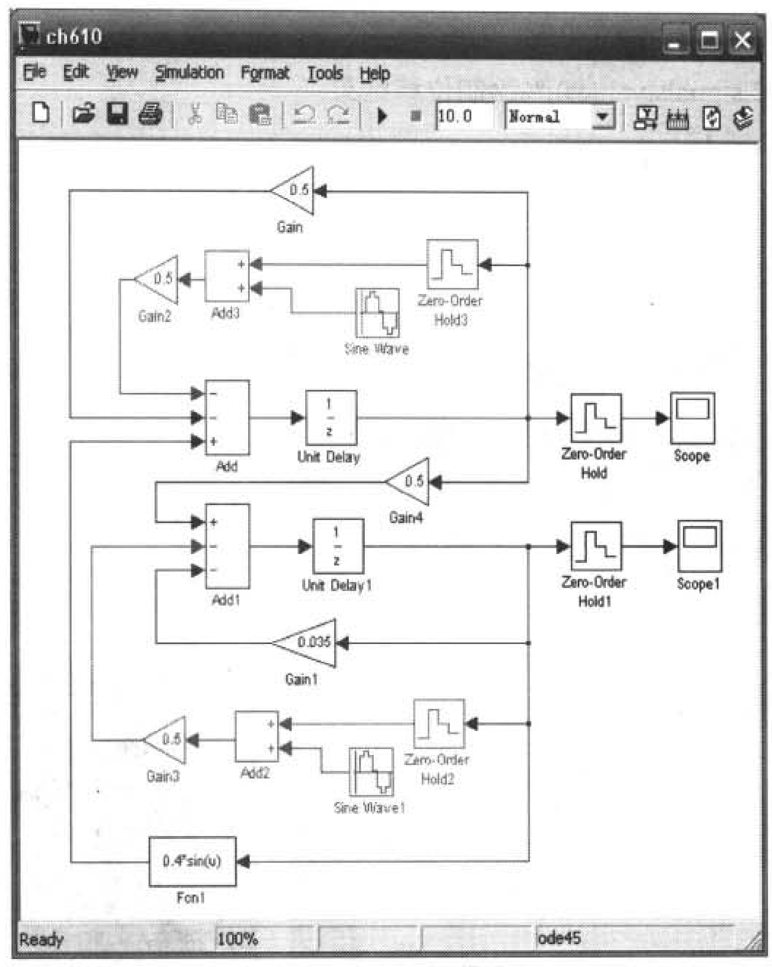


图 6-20 多速率系统模型图

■ 双击 Scope 模块，在弹出的参数对话框中单击 按钮，在弹出的对话框中单击 Data History 选项卡，选中 Save data to workspace 复选框，在 Variable name 文本框中输入 x1，如图 6-21 所示。按同样的方法设置 Scope1 模块，在 Variable name 文本框中输入 x2。

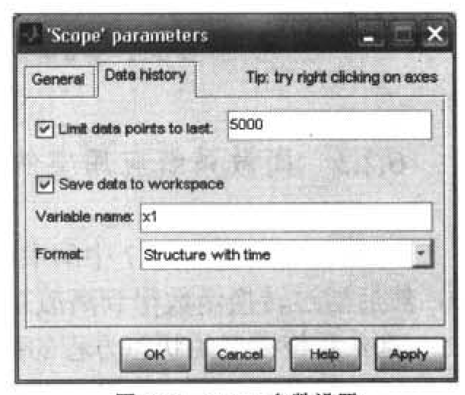


图 6-21 Scope 参数设置

（2）运行仿真。在 MATLAB 命令窗口中输入如下命令：

```
>>ch64
```

其中 ch64.m 文件内容如下：

```
% ch64.m
sim('ch610');%进行系统仿真
figure(1)
plot(x1.time,x1.signals.values);%绘制速度结果
set(gca,'FontSize',12)
xlabel('Time');
ylabel('x1')
title('离散系统')
```

```
grid on
figure(2)
plot(x2.time,x2.signals.values);%绘制位移结果
set(gca,'FontSize',12)
xlabel('Time');
ylabel('x2')
title('离散系统')
grid on
```

（3）运行结果。如图 6-22 和图 6-23 所示。

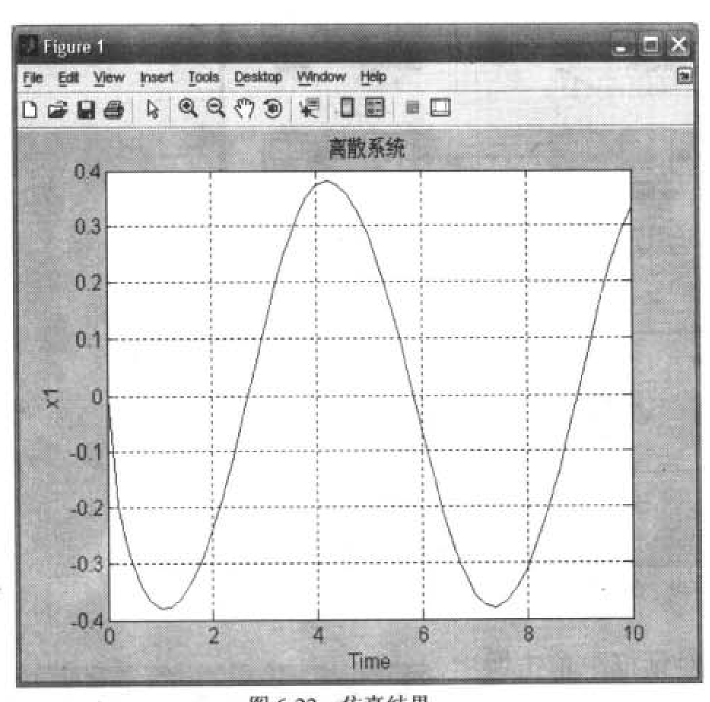


图 6-22 仿真结果

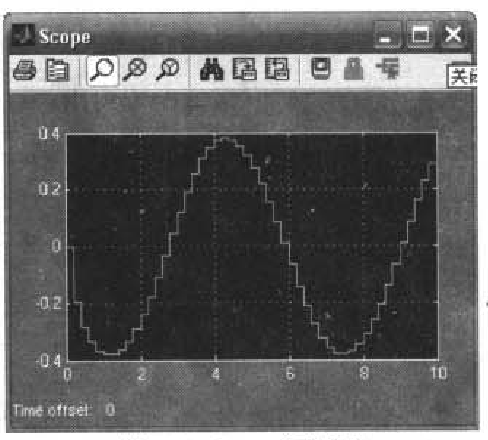


图 6-23 Scope 仿真结果

6.2.3 离散系统应用实例

【例 6.11】依照例 6.7 中的参数建立系统模型的状态方程，确定连续系统矩阵 A、B、C、D，然后通过转换函数得到离散系统模型的状态方程的矩阵 A、B、C、D。

建立系统模型的状态方程的操作步骤如下：

（1）获取离散系统模型的状态方程的矩阵 A、B、C、D，利用函数 c2d。

```
sys=ss(A,B,C,D); %建立连续状态方程
sysd=c2d(sys,0.02,'zoh'); %得到离散状态方程
```

（2）首先建立 Simulink 模型。建立的模型如图 6-24 所示，保存文件名为 ch611.mdl。

- 双击 From Workspace 模块，在弹出的参数对话框中设置 Data 为 EI，单击 OK 按钮。
- 双击 Discrete state-Space 模块，在弹出的参数对话框中，设置 A 为 sysd.a，设置 B 为 sysd.b，设置 C 为 sysd.c，设置 D 为 sysd.d，Initial conditions 为 0，Sample time 为 0.02。如图 6-25 所示，单击 OK 按钮。

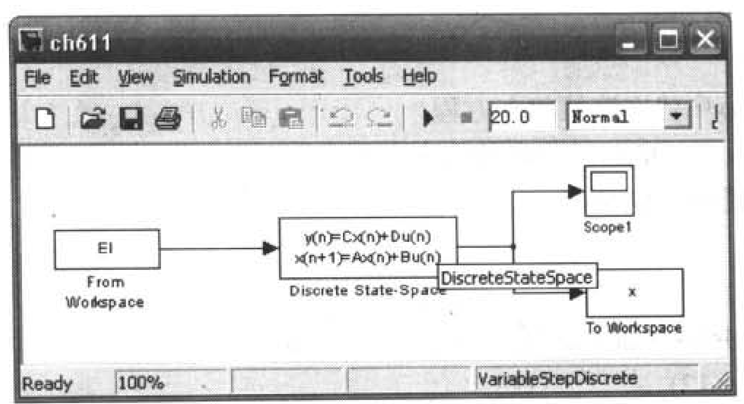


图 6-24　模型图

Function Block Parameters: Discrete State-Space

Discrete State Space

Discrete state-space model:
x(n+1) = Ax(n) + Bu(n)
y(n) = Cx(n) + Du(n)

Main　State properties

A:
sysd.a

B:
sysd.b

C:
sysd.c

D:
sysd.d

Initial conditions:
0

Sample time (-1 for inherited):
0.02

OK　Cancel　Help　Apply

图 6-25　Discrete State-Space 模块参数设置

➢ 双击 To Worksapce 模块，在弹出的对话框中设置 Variable name 为 x，单击 OK 按钮。

（3）运行仿真。编写 M 文件来输入系统仿真参数，运行 Simulink 模型以及查看结果，保存文件名为 ch65.m。

在 MATLAB 命令窗口中输入：

```
>>ch65
```

ch65.m 文件如下：

```
% ch64.m
load('EI.mat')%调入 EI 地震数据
M=2923.38;%设置系统质量
K=1391060;%设置系统刚度
C=6373.74;%设置系统阻尼
A=[0 1;-K/M -C/M];
B=[0;1];
```

```
C=eye(2);
D=zeros(2,1);
sys=ss(A,B,C,D);%建立连续状态方程
sysd=c2d(sys,0.02,'zoh')%得到离散状态方程
sim('ch611');%进行系统仿真

figure(1)
plot(tout,x.signals.values(2:end,1));%绘制速度结果
set(gca,'FontSize',12)
xlabel('Time');
ylabel('Displacement')
title('离散系统')
grid on

figure(2)
plot(tout,x.signals.values(2:end,2));%绘制位移结果
set(gca,'FontSize',12)
xlabel('Time');
ylabel('Velocity')
title('离散系统')
grid on
```

（4）仿真结果。运行上述程序后，会得到如图 6-26 和图 6-27 所示的结果。图 6-26 为结构响应速度曲线，图 6-27 为结构响应位移曲线。

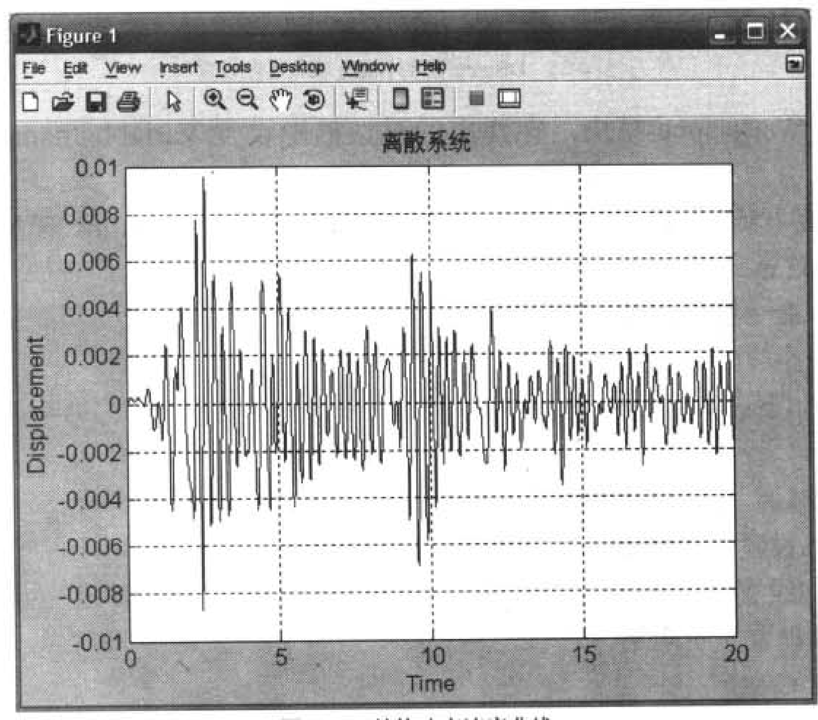


图 6-26　结构响应速度曲线

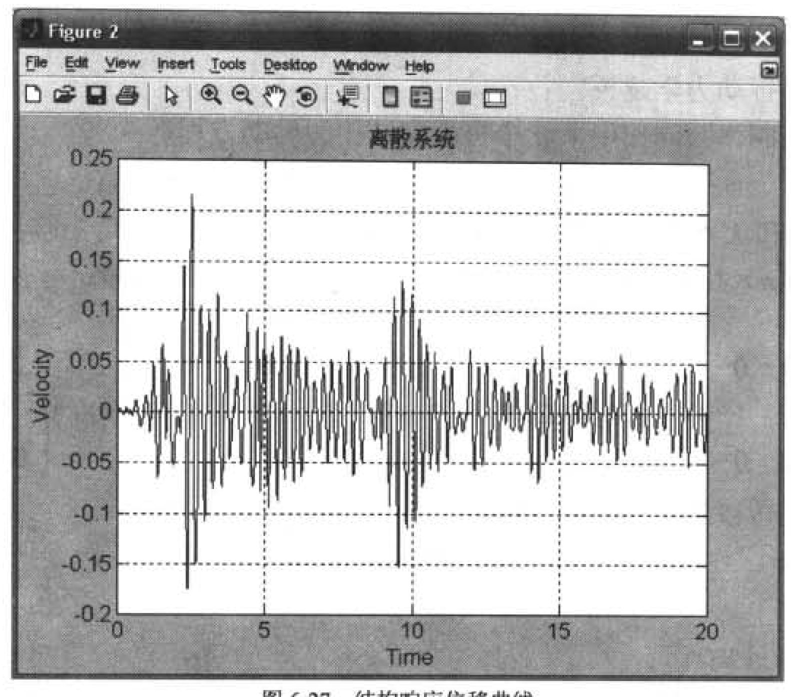


图 6-27 结构响应位移曲线

6.3 离散—连续混合系统

在现代控制系统中，被控系统均为连续的，如建筑结构主动控制中，受控系统建筑结构为一个连续系统，而控制系统为离散子系统。对于这类离散—连续混合系统，模型参数设置中的所有求解器都可以使用。

【例 6.12】 对于一个三层建筑结构，如图 6-28 所示，利用连续系统进行建模，利用离散控制系统进行控制。

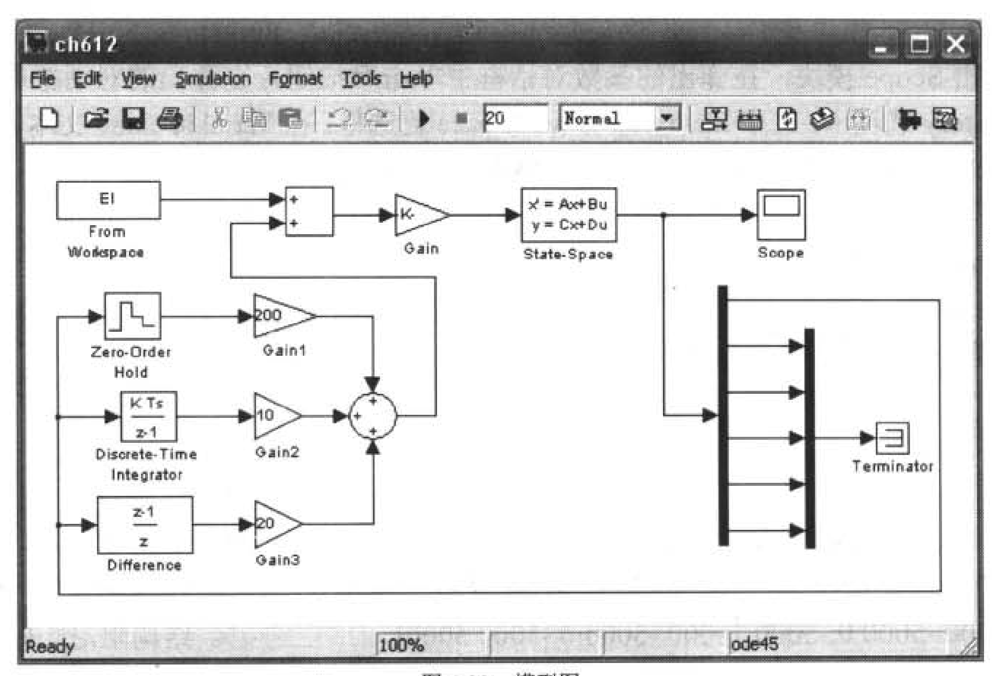


图 6-28 模型图

操作步骤如下：

（1）对系统进行动力学建模。

考虑地震和控制律共同作用下 n 个自由度结构的运动方程：

$$M\ddot{x}(t)+C\dot{x}(t)+Kx(t)=Bu(t)+MI\ddot{g}(t)$$

式中，M、C 和 K 分别为 $n\times n$ 阶的质量矩阵、阻尼矩阵和刚度矩阵；B 为 $n\times r$ 阶控制律位置矩阵；X 为 $n\times 1$ 阶的结构位置向量；u 为 $r\times 1$ 阶控制律向量；$\ddot{g}$ 为地震加速度；I 为 n 维的数向量。

$$M=\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix},\quad K=1.25*10^6*\begin{pmatrix}2&-1&0\\-1&2&-1\\0&-1&2\end{pmatrix},\quad C=5000*\begin{pmatrix}2&-1&0\\-1&2&-1\\0&-1&2\end{pmatrix}$$

（2）运动方程转移到状态空间方程为：

$$\dot{Z}(t)=AZ(t)+DU(t)+F(t)$$

式中，

$$Z(t)=\begin{bmatrix}X(t)\\ \dot{X}(t)\end{bmatrix},\quad A=\begin{pmatrix}0&I\\-M^{-1}K&-M^{-1}C\end{pmatrix},\quad D=\begin{bmatrix}0\\M^{-1}B\end{bmatrix},\quad F(t)=\begin{bmatrix}0\\I\ddot{g}(t)\end{bmatrix}$$

（3）建立 Simulink 模型。本例并不注重控制方法和控制效果，而是为了演示离散一连续混合系统，采用简单的 PID 离散控制系统。

建立模型，如图 6-30 所示，保存为 ch612.mdl，模型参数设置如下：

- Gain 模块中的参数 Gain 设置为[0 0 1]'。
- Gain1 模块中的参数 Gain 设置为 200。
- Gain2 模块中的参数 Gain 设置为 10。
- Gain3 模块中的参数 Gain 设置为 20。
- State-Space 模块设置：A 为 A，B 为 B，C 为 C，D 为 D。
- 双击 From Workspace 模块，在弹出的参数对话框中设置 Data 为 EI。
- Zero-Hold 模块和 Discrete-Time Integrator 模块中的 Sample time 为 0.2。
- 双击 Scope 模块，在弹出的参数对话框中单击按钮，在弹出的对话框中单击 Data History 选项卡，选中 Save data to workspace 复选框，在 Variable name 文本框中输入 x。

（4）运行仿真。在 MATLAB 命令窗口输入如下命令：

```
>>ch66
```

其中 ch66.m 文件内容如下：

```
% ch66.m
% 这是一个多自由度受到地震荷载的情况

clear
load('EI.mat');                                   % 导入地震波
M=[2500 0 0;0 2500 0;0 0 2500];                   % 结构质量矩阵
K=[2500 -1250 0;-1250 2500 -1250;0 -1250 1250]*1000;   % 结构刚度矩阵
C=[10000 -5000 0;-5000 10000 -5000;0 -5000 5000];      % 结构阻尼矩阵
ling=zeros(3);                                    % 定义零矩阵
```

```
danwei=eye(3);                    % 定义单位矩阵
% 转换到状态空间
A=[ling danwei;-K*inv(M) -C*inv(M)];
B=[ling;danwei];
C=[eye(6)];
D=[ling;ling];
sim('ch612');

figure(1)
plot(x.time,x.signals.values(:,1:3));%绘制速度结果
set(gca,'FontSize',12)
xlabel('Time');
ylabel('Displacement');
title('混合系统');
grid on

figure(2)
plot(x.time,x.signals.values(:,4:6));%绘制速度结果
set(gca,'FontSize',12)
xlabel('Time');
ylabel('Velocity');
title('混合系统');
grid on
```

（5）运行结果。图 6-29 所示为 3 个自由度的位移输出结果，图 6-30 所示为 3 个自由度的速度输出结果。

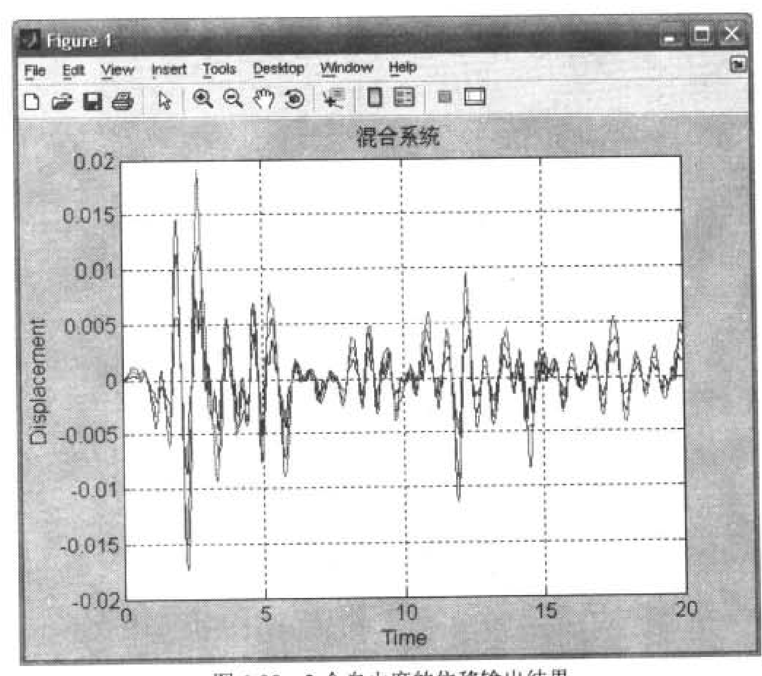


图 6-29　3 个自由度的位移输出结果

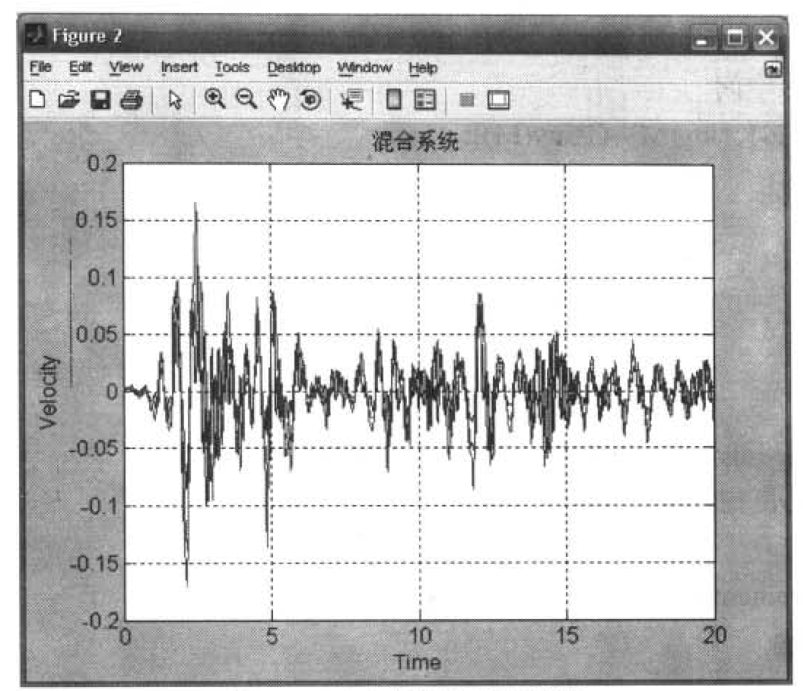


图 6-30 3 个自由度的速度输出结果

第7章 子 系 统

对于简单的系统来说，可以直接建立系统的模型，并分析模块之间的相互关系以及模块的输入输出关系。但是，对一个复杂系统或者一个大系统中存在多个相对独立的子系统来说，Simulink 中将会包含非常多的模块，使得各个模块之间的相互关系显得非常复杂，不利于分析。而子系统正是针对以上原因设计的，可以将联系比较紧密的模块或者归属于一个子系统的模块进行封装，这样就能够对大系统模型一目了然。

本章主要内容包括：

- 子系统的创建
- 子系统的封装
- 条件执行子系统

7.1 子系统的创建

Simulink 提供的子系统功能可以大大地增强 Simulink 系统模型框图的可读性，可以不必了解子系统中每个模块的功能就能够了解整个系统的框架。

子系统可以理解为一种“容器”，我们可以将一组相关的模块封装到这个子系统模块当中，并且等效于原系统模块群的功能，而对其中的模块我们可以暂时不去了解。组合后的子系统可以进行类似模块的设置，在模型仿真过程中可以作为一个模块。

7.1.1 创建自己的子系统

下面来介绍两种建立子系统的方法。

1. 在已有的系统模型中建立子系统

设已有 Simulink 模型图如图 7-1 所示，选择框选需要封装的模块区域（用 Shift 键和鼠标左键配合可以达到同样的目的），框选如图 7-2 所示的区域。然后右键单击，弹出浮动菜单，

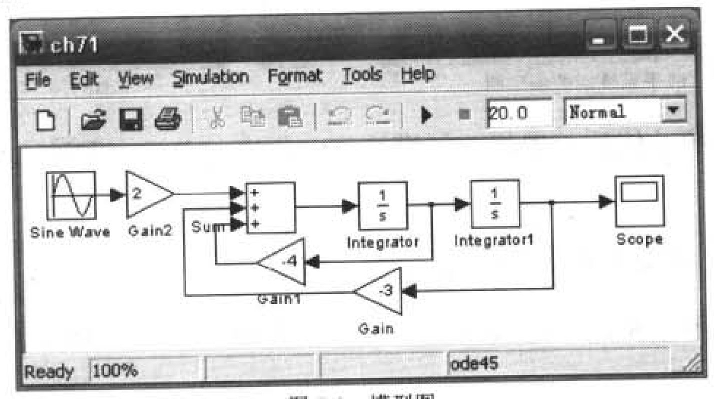


图 7-1 模型图

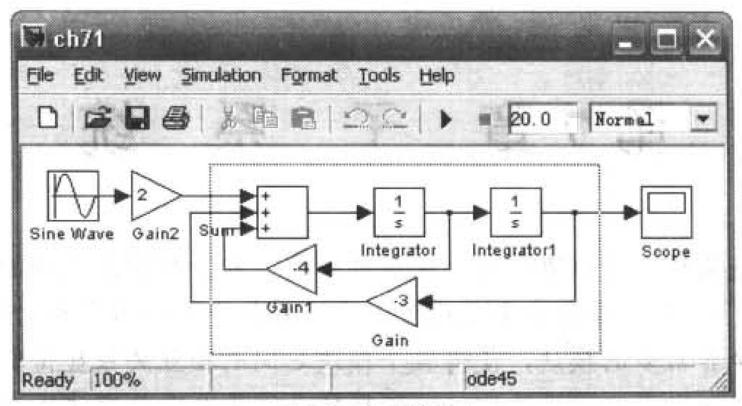


图 7-2　框选模型图

选择 Create Subsystem 命令，如图 7-3 所示。创建子系统的结果如图 7-4 所示。然后双击 Subsystem 模块，弹出子模型如图 7-5 所示。

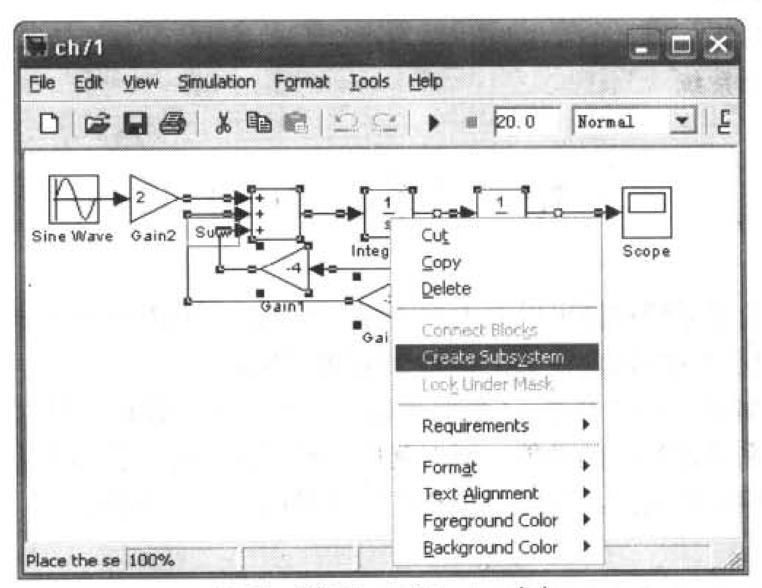


图 7-3　选择 Create Subsystem 命令

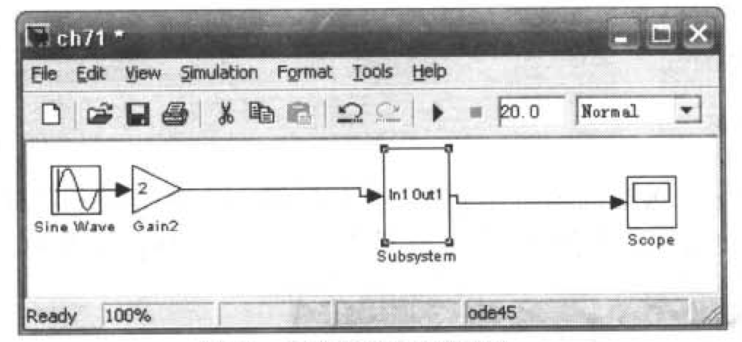


图 7-4　生成子系统后的模型图

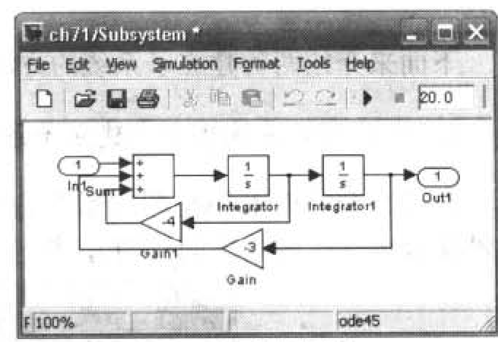


图 7-5　子系统模型图

很明显，子系统的功能就是把相关的模块集中起来，并没有删除，这个系统的结构仍然没有变。

2. 在已有的系统模型中新建子系统

简单建立模型如图 7-6 所示，双击 Subsystem 模块，然后在弹出的窗口中建立如图 7-7 所示的模型。

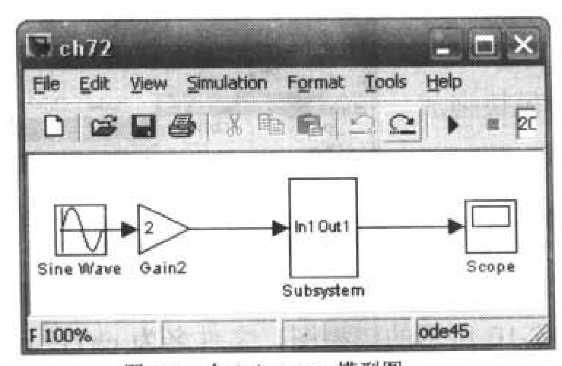



图 7-6 含 Subsystem 模型图

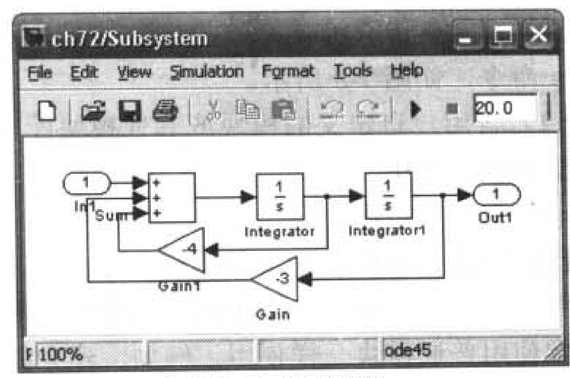


图 7-7 子系统模型图

这两种创建子系统最后实现的是一模一样的功能，只不过操作顺序不同。前者是先将这个结构搭建起来，然后把相关模块封装起来。后者则是先做一个封装容器，然后在封装容器中添加模块。

对于一个相对简单的模型，我们采用第一种，这种操作一般不容易出错，能够顺利搭建模型。而对于非常复杂的系统，我们事先将模型分成若干个子系统，然后再采用第二种方法进行建模。

在使用 Simulink 子系统建立系统模型时有如下几个常用操作：

- 子系统命名：命名方法与模块命名类似，是用有代表意义的文字来对子系统进行命名，有利于增强模块的可读性。
- 子系统的编辑：用鼠标双击子系统模块的图标，打开子系统并对其进行编辑。
- 子系统的输入：使用 Sources 模块库中的 Inport 输入模块，即 In1 模块，作为子系统的输入端口。
- 子系统的输出：使用 Sinks 模块库中的 Outport 输出模块，即 Out1 模块，作为子系统的输出端口。

7.1.2 用子系统来自定义库

前面介绍了 Simulink 子系统的封装技术。子系统的封装给不同领域的设计者带来了很大的方便，使用者调用这些模块如同调用 Simulink 内部模块一样，可以使用封装子系统的帮助来了解如何进行模块设置，而不需要了解模块内部的构成。但是，如果封装的子系统比较多，应用的范围也不相同，如何有效地管理这些模块就成为一个非常重要的问题。

模块库就是指具有某种属性的一类模块的集合。在 Simulink 模块浏览器中有大量的模块库，用户就可以调用其中的模块来进行各种系统的仿真。

Simulink 允许用户自己开发模块，并在建立自己模块库的同时对开发的模块进行有效的管理。自定义模块库的使用方法和 Simulink 其他自带的模块库的使用完全一样。

- 模块库：某一类模块的集合。
- 库模块：模块库中的模块。
- 引用块：调用模块库中的模块。
- 关联：引用块与对应模块库中的模块建立关联，以便在模块库中的模块发生改变时，Simulink 模块库中的模块会自动更新。

在使用模块库之前，首先要创建模块库，其操作步骤如下：

（1）在 Simulink Library Browser 窗口中选择 File|New|library 命令，如图 7-8 所示。

（2）将用户自定义的模块或其他模块库中的模块拖放到新建的模块库中。

（3）保存模块库。

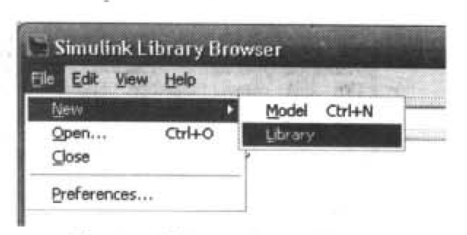


图 7-8 选择 File|New|library 命令

【例 7.1】创建模块库和使用模块库。

将上一节中创建好的模块拖放到新建的模块库中，如图 7-9 所示，保存为 ch711.mdl，然后就像使用普通模块一样使用。例如，建立如图 7-10 所示的模型图，文件名为 ch712.mdl，可以发现运行结果与上节的结果一致。

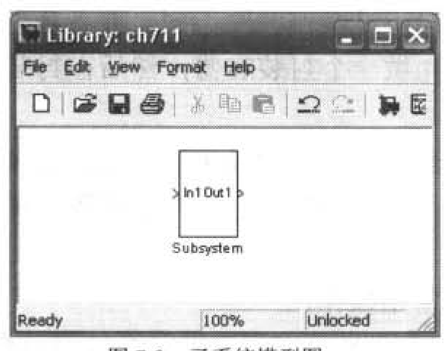


图 7-9 子系统模型图

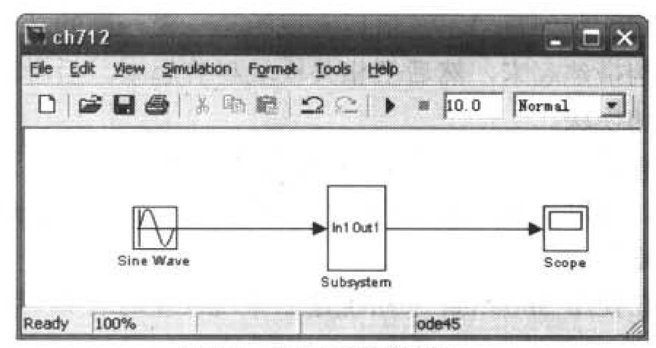


图 7-10 使用子系统模型图

7.2 子系统的封装

在前面介绍过通过一个子系统，可以方便模型仿真和分析。然而，仿真以前，必须对子系统中的模块进行初始化，在特殊的情况下，一个大系统构造成的子系统往往需要不断地调整参数，如果每次都是打开子系统窗口进行参数设置，必将为仿真和分析带来很多麻烦。

在模型中，子系统的功能就相当于一个普通的模块，具有特定的输入端口和输出端口。

在这里要分清楚封装子系统和建立子系统是两个不同的概念，分别介绍如下：

建立子系统是将一组完成相关功能的模块包含到一个子系统当中，用一个模块来表示，主要是为了简化 Simulink 模型，增强 Simulink 模型的可读性，便于我们仿真和分析。在仿真前，需要打开子系统模型窗口，对其中的每个模块分别进行参数设置。虽然增强了 Simulink 模型的可读性，但并没有简化模型的参数设置。当模型中用到多个这样的子系统，但是每个子系统中模块的参数设置都不相同时，这就显得很不方便，而且容易出错。

为了解决简单建立子系统的不足，我们可以对子系统进行封装。将完成特定功能的相关模块集合在一起，对其中经常要设置的参数设置为变量，然后封装，使得其中的变量可以在封装系统的参数设置对话框中统一进行设置，这就大大简化了参数的设置，而且不容易出错。这非常有利于进行复杂的大系统仿真。

封装后的子系统可以作为用户的自定义模块，和普通模块一样添加到 Simulink 模型中应用，也可添加到模块库中以供调用。封装后的子系统可以定义自己的图标、参数和帮助文档，完全与 Simulink 的其他普通模块一样。双击封装子系统模块，弹出对话框，进行参数设置，如果有任何问题，可以单击 help 按钮，不过这些帮助都是要创建者自己进行编写的。

下面对 Simulink 封装子系统的几个特点进行总结：

- 可以自定义封装子系统的图标。
- 双击封装后的子系统，弹出参数对话框，其中对话框是自定义的。
- 封装子系统的帮助文档都是自定义编写的。
- 封装子系统有自己的工作区域。

以上特点为模型设计带来了很大的方便，具体如下：

- 将子系统作为一个黑匣子，用户不必了解其中的具体细节而可以直接使用。
- 将子系统中模块的参数设置统一到一个参数对话框，大大方便了系统的参数设置。
- 保护知识产权，防止篡改。

7.2.1 子系统封装示例

下面通过一个实际例子来说明如何封装一个子系统。

【例 7.2】封装子系统。

（1）建立模型文件名为 ch77.mdl，如图 7-11 所示，并对需要封装的模块进行框选。

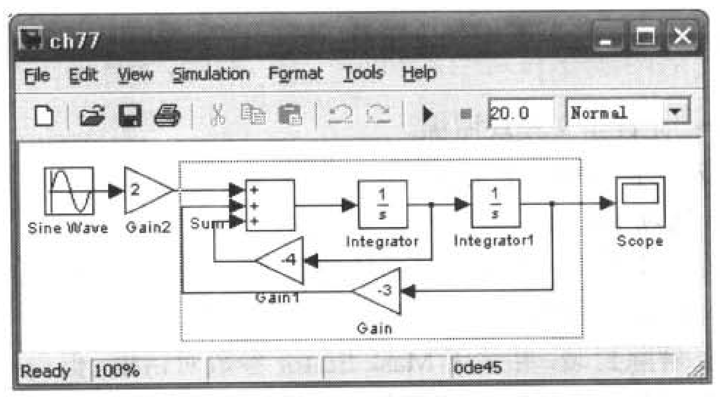


图 7-11 模型图

（2）生成子系统，右击选中的模块，在弹出的菜单中选择 Create Subsystem 命令，并将文件另存为 ch78.mdl，如图 7-12 所示。

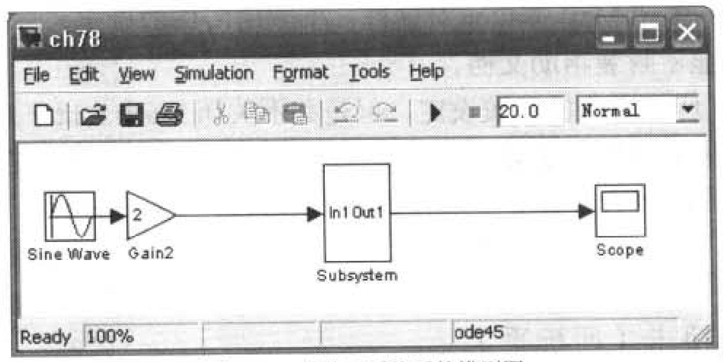


图 7-12 封装子系统后的模型图

（3）封装子系统，右击生成的 Subsystem 模块，在弹出的菜单中选择 Mask Subsystem 命令，弹出如图 7-13 所示的对话框。在其中可以进行各种设置，这将在后面进行介绍。

Mask Editor 参数对话框可以创建和编辑封装子系统。右击生成的 Subsystem 模块，在弹出的菜单中选择 Mask Subsystem 命令，会弹出 Mask Editor 参数对话框。

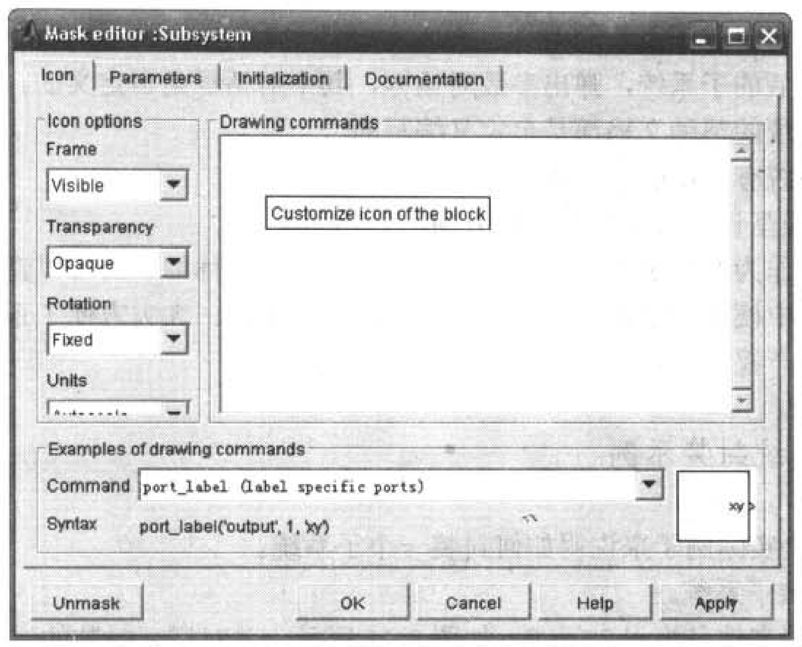


图 7-13 Mask Editor 对话框

Mask Editor 参数对话框包含了 4 个选项卡，每个选项卡都可以定义封装（mask）的一个特性：

（1）Icon 选项卡：允许定义模块图标。

（2）Parameters 选项卡：允许定义和描述封装对话框和参数对的字符变量。

（3）Initialization 选项卡：允许指定初始化命令。

（4）Documentation 选项卡：允许定义封装的类型，并且设定模块的描述和帮助。

（5）Mask Editor 参数对话框中最下面 5 个按钮的功能。

- Unmask 按钮：解除封装，并关闭 Mask Editor 参数对话框，但是封装的信息仍然保留。为了恢复封装，右击选择的模块，在弹出的菜单中选择 Create Mask 命令。将弹出 Mask Editor 对话框，并显示以前的设置。当模型被关闭后，其中的封装信息就被清除了。
- OK 按钮：应用所有设定，并关闭 Mask Editor 对话框。
- Cancel 按钮：关闭 Mask Editor 对话框，不应用所作的设定。
- Help 按钮：显示封装帮助文档。
- Apply 按钮：应用所作的参数设定，但是并不关闭 Mask Editor 对话框。

为了能够查看没有封装的子系统，可以右键单击子系统，然后在弹出的菜单中选择 LookUnder Mask 命令，将会打开子系统，而且模块封装不会受影响。对每个选项卡的介绍如下。

7.2.2 Icon 选项卡（图标页）

Icon 选项卡可以创建块图标，在图标中可包含描述性文字、状态方程、图像和图形，如图 7.33 所示。

（1）Drawing commands 绘制命令

用户可以在 Drawing commands 区编写绘制块图标的命令。在 Simulink 中提供了一套命

令，可以绘制文本、一个或多个图，可以显示传递函数，可以查看 Icon 选项卡中的在线帮助命令实例。用户只能通过这些命令来绘制图标。Simulink 就会按顺序执行命令区的命令，绘制命令有权调用所有封装工作空间中的变量。

下面这段命令演示了如何为 mx+b 封装子系统创建一个图标。

```
pos=get_param(gcb, 'Position');
width=pos(3)-pos(1);height=pos(4)-pos(2);
x=[0， width];
if(m>= 0),y=[0,(m*width)]; end
if(m<0),y=[height, (height+(m*width))];end
```

这些初始化命令定义了绘制图形的数据，这些数据不受模块形状的影响而精确地绘制图标。绘制图标的命令是 plot(x,y)。

（2）Examples of drawing commands 绘制命令的实例

该选项组说明不同的 Simulink 支持的图标绘制命令。为了能够了解其中命令的语法，可以从命令下拉菜单中选择相应的命令，Simulink 就会显示所选择命令的实例，并会在右下角产生一个图标。

（3）Iconoptions 控制图标显示选项组

这个区域可以控制以下几个方面的显示：

- 框架（Frame）：图标框架是将模块封闭起来的矩形。用户可以通过选择 Frame 下拉列表框中的 Visible 或者 Invisible 选项来决定显示还是隐藏框架。默认情况是显示框架。
- 透明（Transparency）：可以将图标设置成为不透明的或者透明的，也就是说，是否显示图标下面的内容。默认设置是不透明，即覆盖 Simulink 绘制出来的信息，如端口标签。
- 旋转（Rotation）：当模块旋转或者空翻（就是上下端口不变，左右端口变化）时，可以设定图标是否也跟着变化还是保持方向不变。默认情况下是不随模块的旋转而变化。
- 单位（Units）：这个选项控制了绘制命令坐标系统的单位，仅对 plot 和 text 绘制命令有效，可以选择 3 个选项之一：Autoscale，Normalized 和 Pixel。
  - Autoscale：可以根据框架来适当地调整图标的大小，当模块调整大小时，图标也同时调整大小。
  - Normalized：在模块框架内绘制图标，左下角坐标是（0，0），右上角坐标是（1，1），坐标 X 和 Y 只能在 0～1 之间取值，当模块调整大小时，图标也同时调整大小。
  - Pixel：用 X 和 Y 的像素值来绘制图标，这种绘制图形的方法，在模块进行大小调整时，图标不会改变大小。如果要强制随模块的变化来调整大小，必须根据模块的大小来定义绘制命令。

7.2.3 Parameters 选项卡

Parameters 选项卡可以创建和修改封装子系统的参数，这些参数决定了封装子系统的各种特性和行为，如图 7-14 所示。

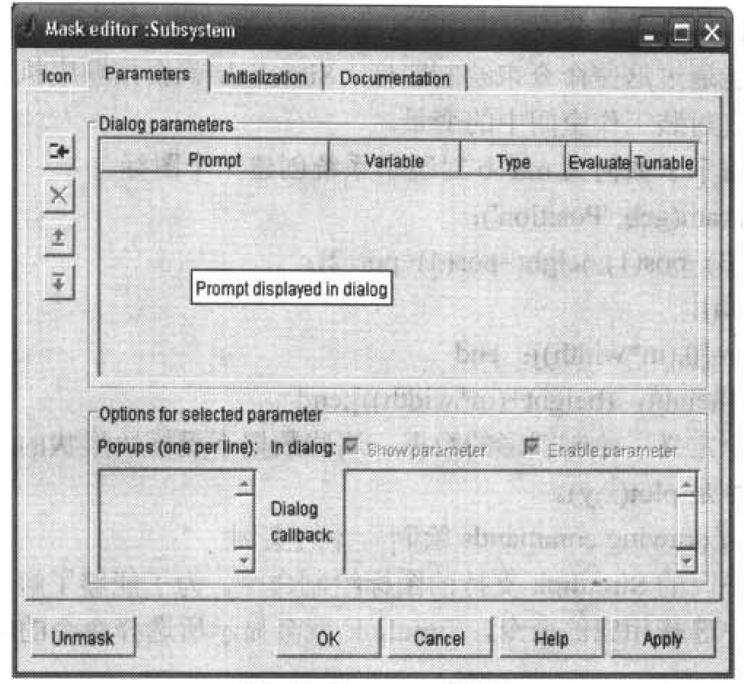


图 7-14　Parameters 选项卡

Parameters 选项卡包括两个选项组：

- Dialog parameters 选项组：可以选择和改变一些主要封装子系统的特性。
- Options for selected parameter 选项组：可以为 Dialog parameters 选项组中设定的参数添加附加的选项。

Parameters 选项卡的左端按钮允许用户添加、删除封装子系统的参数和改变这些参数的位置。

（1）Dialog parameters 控制面板

以表格的形式列举封装子系统的参数。每一行显示一个封装子系统参数的主要特征。例如，进行如图 7-15 所示的设置，然后单击 OK 按钮，最后双击封装子系统 Subsystem 模块，就会弹出如图 7-16 所示的对话框。

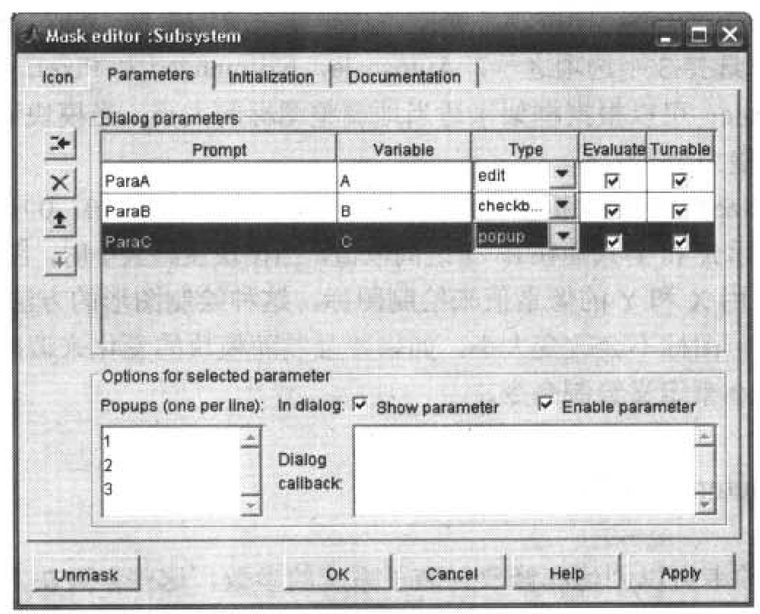


图 7-15　Parameters 选项卡

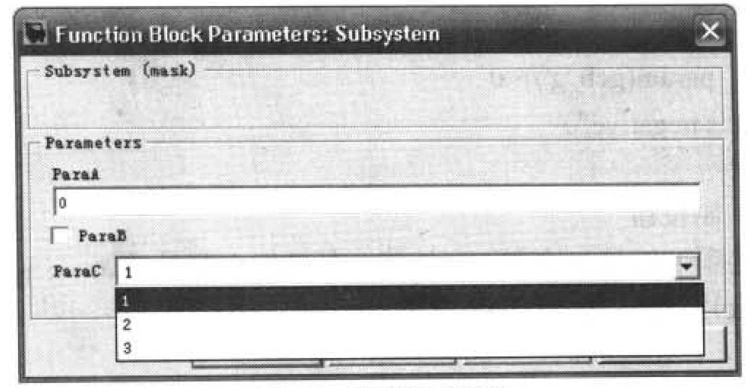


图 7-16 子系统参数设置

下面对图 7-15 所示的 Parameters 选项卡中的参数进行说明。子系统参数设置如图 7-17 所示。

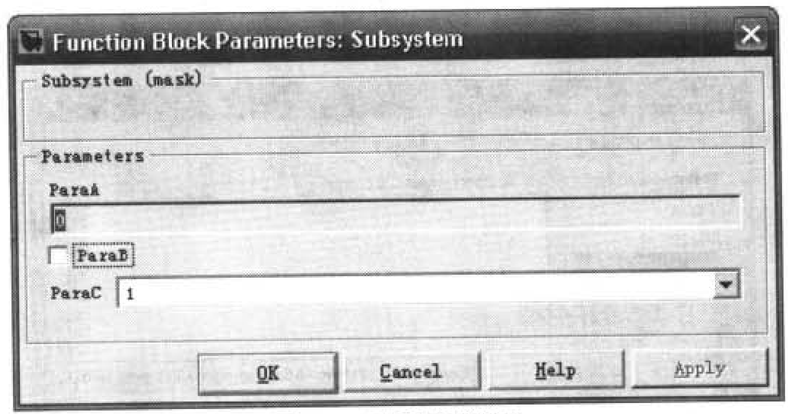


图 7-17 子系统参数设置

- Type：指编辑参数的方式，包括 edit、checkbox 和 popup 形式，分别为文本输入、多选和下拉菜单形式。
- Evaluate：如果选中该复选框,表示在将用户输入的表达式赋予相应变化的变量之前对表达式求值。否则，Simulink 就将表达式作为字符值赋予变量。例如，如果用户在编辑框中输入 gain，而且 Evaluate 是选中的，那么 Simulink 就会先求出 gain 的值，然后赋给相应的变量。假如没有选中 Evaluate，则直接给字符串 'gain' 赋予变量。
- Tunable：该选项允许在 Simulink 进行仿真的过程中改变封装子系统中的参数。

（2）Options for selected parameter 选项卡

该选项卡允许用户向 Dialog parameters 表中的参数添加附加的选项。

- Show parameter 复选框：只有选中了，所选择的参数才会出现在封装子系统模块的参数对话框中。
- Enable parameter 复选框：不选中该复选框，所选择的参数在封装子系统模块的参数对话框中是不可编辑的。例如，当选择 ParaA 菜单选项，然后不选中该复选框，单击 OK 按钮，最后双击封装子系统模块，就会弹出参数对话框，如图 7-44 所示，注意其中的 Mass 参数是不可编辑的。
- Popup：仅当 Dialog Parameters 控制面板中的编辑方式 Type 为 pop-up 时，才会处于可编辑状态。输入相应的 pop-up 菜单选项，一行一个菜单命令。
- Callback（回调）：在用户编辑所选择的菜单时，才允许 Simulink 执行所输入的代码。回调能够在模块的工作范内创建和引用变量。如果回调需要封装子系统的参数值，可

以使用 get_param 去获得这个值。例如：

```
if str2num(get_param(gcb,'g'))<0
  error（'Gain is negative.'）
end
```

（3）参数增减控制按钮

在 Parameters 选项卡的最左边有一列按钮，其功能主要是增加、删除参数和对参数进行排序。如图 7-18 所示。

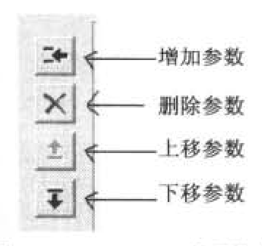


图 7-18　Parameters 选项卡的一列按钮

7.2.4　Initialization 选项卡

Initialization 选项卡允许用户输入 MATLAB 命令来初始化封装子系统，Initialization 选项卡如图 7-19 所示。

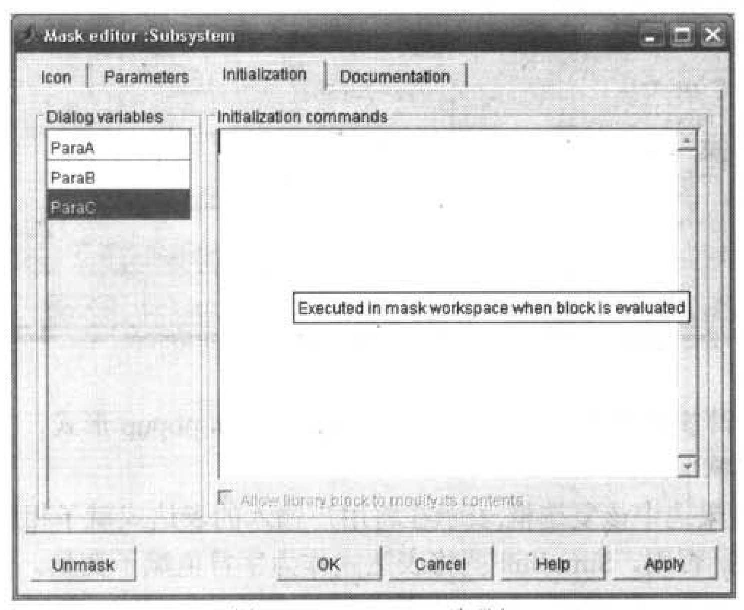


图 7-19　Initialization 选项卡

Simulink 在以下几种情况下执行初始化命令：

- 装载模型。
- 开始仿真和更新模块图形（选择 Edit | UpdateDiagram 命令）。
- 旋转模块。
- 重画模块图标（如果封装子系统的图标代码中有依靠 Initialization 选项卡中初始化的变量）。

（1）Initialization 选项卡

包括以下几个控制选项：

- Dialog variables 选项组：此列表中显示了与封装子系统参数相关的变量名。用户可以从这个列表中复制参数名到 Initialization commands 框中。也可以使用这个列表来更改参数变量，用鼠标双击相应的变量就可以更改了，然后按 Enter 键确定。
- Initialization commands 选项组：在 Initialization commands 中输入初始化命令，可以是任何的 MATLAB 表达式，例如 MATLAB 函数、运算符和在封装模块空间中的变量，但是初始化命令不能够是基本工作空间的变量。初始化命令要用分号来结尾，避

免在 MATLAB 命令窗口中出现回调结果。

- Allow library block to modify contents 复选框：该复选框仅当封装子系统存在于模块库中才可用。选中这个复选框，允许模块的初始化代码修改封装子系统的内容。例如可以允许初始化代码增加和删除模块，还可以设置模块的参数。否则，当试图通过模块库中的模块修改模块中的内容时，Simulink 仿真就会出现错误。不过这个还可以在 MATLAB 命令窗口中实现，选中要修改内部模块的封装子系统模块，然后在命令窗口中输入如下命令：

```
>>set_param(gcb,'MaskSelfModifiable','On');
```

然后保存这个模块。

（2）调试初始化命令

用户可以通过以下几种方法来调试初始化命令。

- 在命令的结尾不是用分号，以便能够在MATLAB命令窗口中直接查看相关命令运行结果。
- 在命令中间设定一些键盘控制命令，如中断、键盘输入参数等，可以实现人机交互，这样就可以清楚地了解每一步运行的结果。
- 可以在 MATLAB 命令窗口中输入以下命令：

```
>>dbstop if error
>>dbstop if warning
```

这些命令可以在初始化命令发生错误时就停止执行，然后用户就可以检查封装子系统的工作空间。

7.2.5 Documentation 选项卡

Documentation 选项卡允许用户定义或修改类型、描述以及封装子系统模块的帮助文档。下面就是 Mx”+Dx’+Kx 的一个实际例子，如图 7-21 所示，然后单击 OK 按钮，最后双击封装子系统模块，弹出对话框如图 7-22 所示。

（1）仔细观察图 7-20、图 7-21 和图 7-22，就会发现图 7-20 中各个参数的作用：

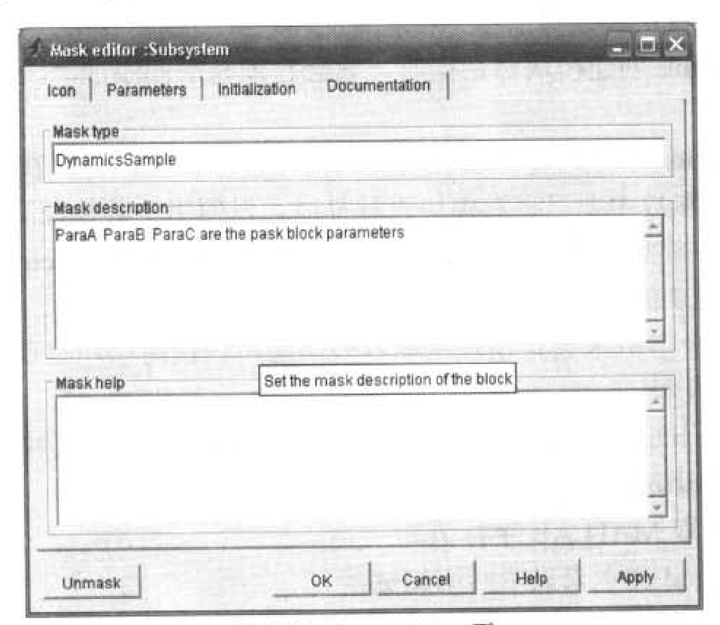


图 7-20　Documentation 页

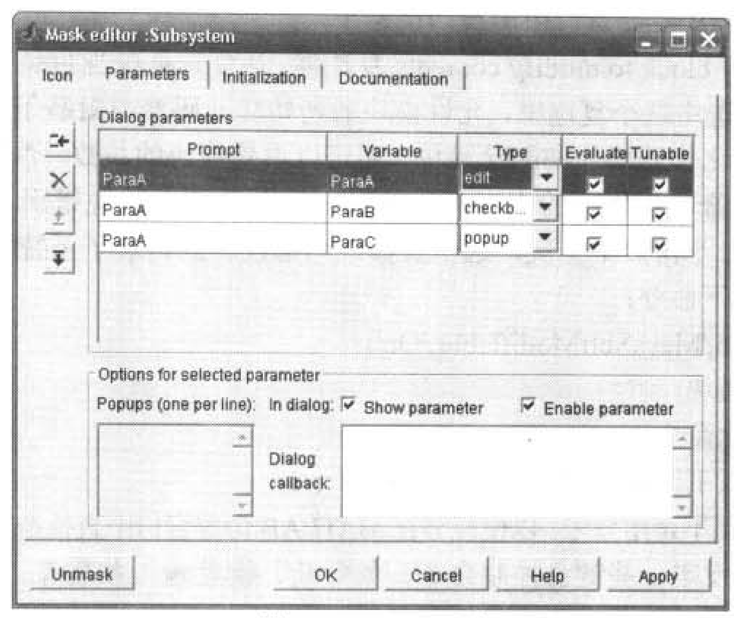


图 7-21　Parameters 页

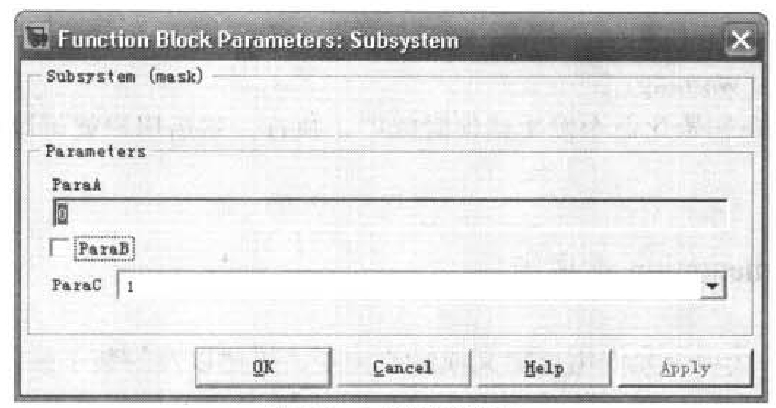


图 7-22　子系统参数设置

- Mask type：为封装子系统模块参数对话框设置说明的标题，仅仅用来对文档进行分类，它出现在模块参数对话框中和所有的模块封装编辑窗中。用户可以选择喜欢的名称，当 Simulink 创建模块对话框时，它会自动在后面添加"(mask)"，以区别系统内建的模块。
- Mask description：对封装子系统模块的工作进行说明，可以在说明中使用 Enter 和 Space 键。此参数中要尽量对模块进行描述，以便用于其他模型当中。
- Mask help：该模块的帮助文档，可以单击封装子系统中的 Help 按钮进行查看。可以讲述如何对模块的参数进行设置，以便用于其他模型当中。

（2）除了使用这个参数来设定模块帮助以外，还可以使用用户自己编写的其他文档。可以使用的文档有以下几种：

- 超链接（如以 http:，WWW，file:，ftp: 和 mailto: 开头的字符）。
- 网络命令（调用浏览器）。
- eval 命令（求取 MATLAB 字符值）。
- 用 HTML-tagged 文本来显示网络浏览器。

Simulink 检查封装子系统模块帮助文档的第一行，如果其中有超链接，如，http://www.

SciEi.com 或者 file:///c:/mydir/helpdoc.html，Simulink 将在浏览器中显示这些内容。

如果 Simulink 检查到有网络命令，例如，web([docroot,'/My Blockset Doc/'get_param(gcb,, MaskType,)',html'])，或者有 eval 命令，例如，eval('!Word My_Spec.doc')，Simulink 就会执行这些命令。否则，Simulink 只会显示模块帮助域中的内容。

7.2.6 联系封装子系统的参数与子系统中的模块参数

在 7.2.3 节中已经讲解了如何设定封装子系统的参数，但是这些参数并没有和子系统内部的模块连接起来。也就是说，即使封装子系统的参数设定完成后，封装子系统模块还是不能够使用，不能和内部的模块进行数据传输。

为了能够使得封装子系统中设置的参数能够被子系统内部的模块所使用，必须将它们连接起来。这就能够让用户使用封装子系统的参数去设置子系统内部的模块参数。为了能够连接这些参数，首先打开模块参数对话框，并在参数对话框中输入所需要的表达式，表达式中的变量都来自于封装子系统所设置的参数。用户可以用封装子系统模块的初始化代码将封装子系统的参数间接地与模块参数连接起来，通过这种方法，初始化代码在封装子系统工作空间创建变量，这些变量的值就是封装子系统的参数的函数，这样就可以通过设置封装子系统参数来获取变量的值，然后通过变量传递给封装子系统内部的模块。

7.3 条件执行子系统

子系统的最基本目的就是将一组相关的模块包含到一个模块中，用以简化系统，使得系统的分析更加容易。例如在一个控制系统中，受控系统就可以视为一个子系统，控制器可以作为一个子系统。从前面的介绍可以发现，这些子系统就和其他一般模块一样，都是具有特定输入输出的模块。对于子系统输入的信号，会产生一个特定的输出信号。但是对于某些特殊情况，并不是所有的输入信号都要产生输出信号，只有在某些特定的条件下才会产生输出信号，这就需要输入一个控制信号。控制信号由子系统模块的特定端口输入，这样的子系统称为条件执行子系统。在条件执行子系统中，输出信号取决于输入信号和控制信号。

根据不同的控制信号，可将条件执行子系统分为如下几类。

（1）使能子系统：当控制信号为正时，子系统开始执行子系统。

（2）触发子系统：当控制信号符号发生变化时，执行子系统。具体有以下形式：

- 控制信号上升沿触发；
- 控制信号下降沿触发；
- 控制信号的双边沿触发。

（3）函数调用子系统：控制信号是由自定义的 S 函数发出调用信号，开始执行信号。

7.3.1 触发子系统及其实例

触发子系统就是在控制信号符号发生变化时执行子系统。下面分别介绍其中的 3 个子系统：

- 控制信号上升沿触发。
- 控制信号下降沿触发。
- 控制信号的双边沿触发。

【例 7.3】 触发子系统建模实例演示。

在本例中同时使用了上述的 3 个子系统（控制信号上升沿触发、控制信号下降沿触发、控制信号的双边沿触发），从中可以发现 3 个子系统之间的区别，模型如图 7-23 所示。

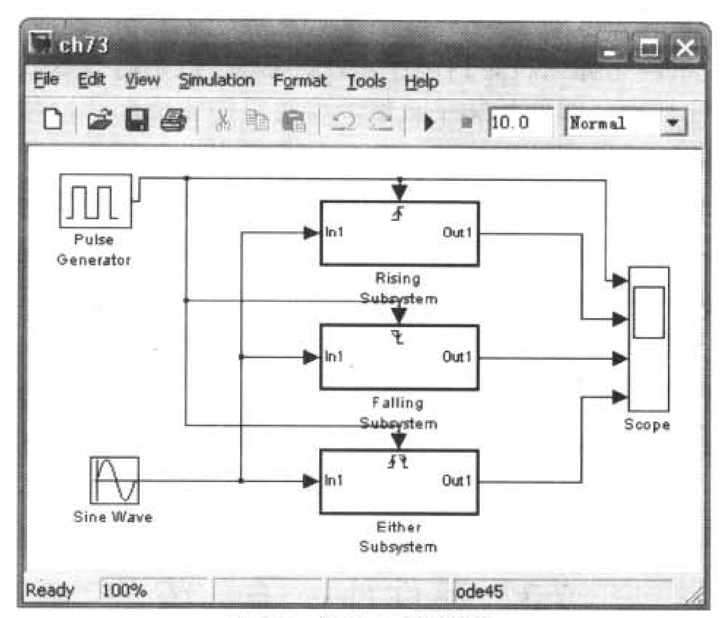


图 7-23 触发子系统模型图

1. 进行模块参数设置

（1）Pulse Generator 模块参数设置。其中 Amplitude 为 1，Period 为 1，Pulse Width 为 50，Phase delay 为 0，如图 7-24 所示。

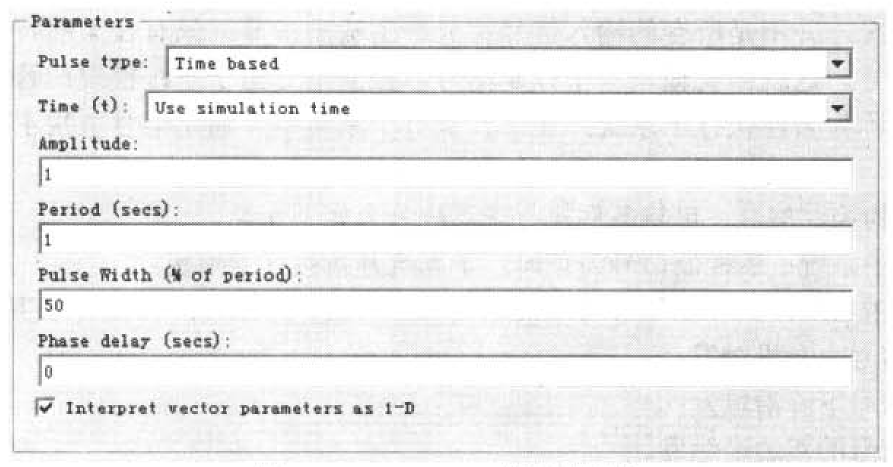


图 7-24 Pulse Generator 模块参数设置

（2）Sine Wave 模块参数设置。其中 Amplitude 为 1，Bias 为 0，Frequency 为 1，Phase Delay 为 0，Sample time 为 0，如图 7-25 所示。

（3）Scope 模块参数设置。双击次模块，在弹出窗口单击按钮，在弹出的参数对话框中设置 Number of axes 为 4。

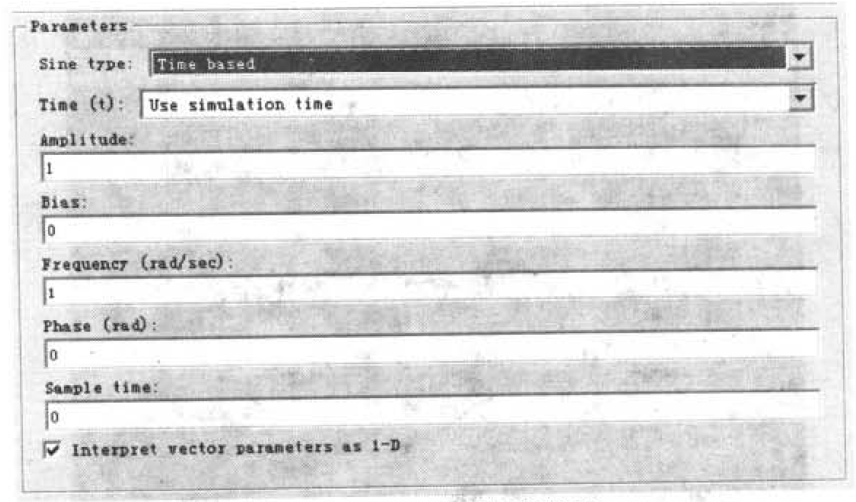


图 7-25　Sine Wave 模块参数设置

（4）Rising Subsystem 模块参数设置。双击此模块，弹出窗口如图 7-26 所示，然后双击其中的 Trigger 模块，在弹出的参数对话框中设置 Trigger Type 栏选项为 rising，然后单击 OK 按钮。

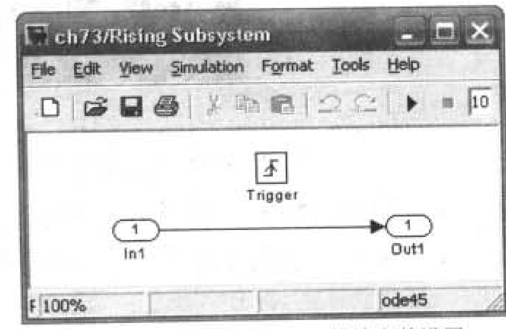


图 7-26　Rising Subsystem 模块参数设置

（5）Falling Subsystem 模块参数设置。双击此模块，弹出窗口如图 7-27 所示，然后双击其中的 Trigger 模块，在弹出的参数对话框中设置 Trigger Type 栏选项为 falling，然后单击 OK 按钮。

（6）Either Subsystem 模块参数设置。双击此模块，弹出窗口如图 7-28 所示，然后双击其中的 Trigger 模块，在弹出的参数对话框中设置 Trigger Type 栏选项为 either，然后单击 OK 按钮。

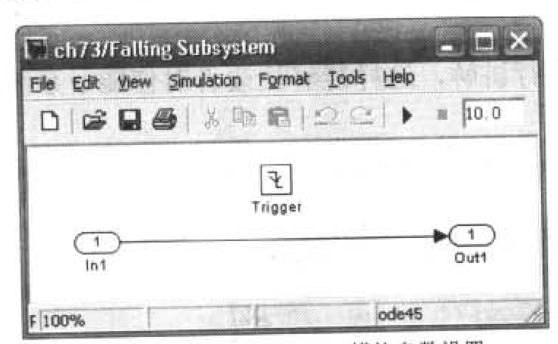


图 7-27　Falling Subsystem 模块参数设置

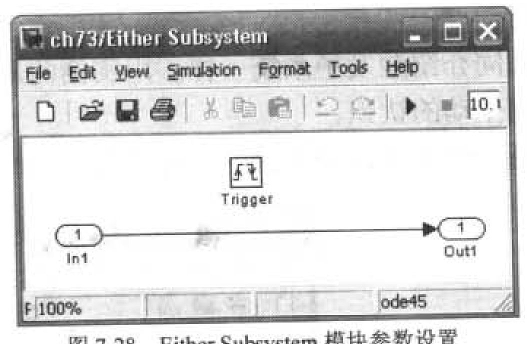


图 7-28　Either Subsystem 模块参数设置

2. 运行仿真

结果如图 7-29 所示。

3. 结果分析

从结果图中可以看出，第 1 幅图是触发信号，第 2 幅是上升沿触发子系统的运行结果，第 3 幅是下降沿触发子系统的运行结果，第 4 幅是双边沿触发子系统的运行结果。

从中可以看出，在触发信号由 1→0 时，下降沿触发子系统和双边沿触发子系统被执行，当 0→1 时，上升沿触发子系统和双边沿触发子系统被执行。

可以发现：

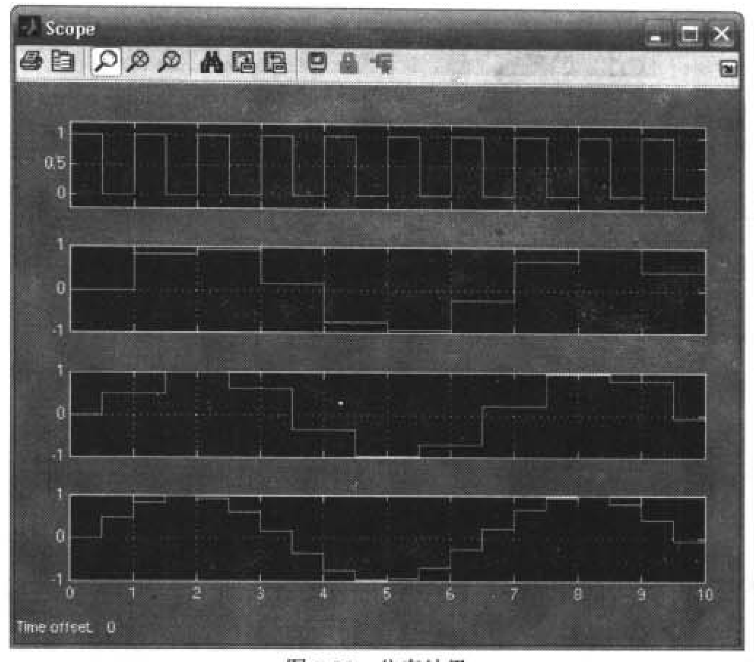


图 7-29　仿真结果

- 上升沿触发子系统在控制信号 0→1 的时候执行。
- 下降沿触发子系统在控制信号 1→0 的时候执行。
- 双边沿触发子系统在控制信号符号发生变化的时候就执行。

7.3.2　使能子系统及其实例

使能子系统是指输入信号为真时，子系统开始执行，也即输入信号为正数时，子系统开始执行。

如同介绍触发子系统一样，用实例的形式进行讲解，这样能够让读者对使能子系统有更加清晰生动的认识。

【例 7.4】 使能子系统的文件名为 ch74.mdl，如图 7-30 所示。

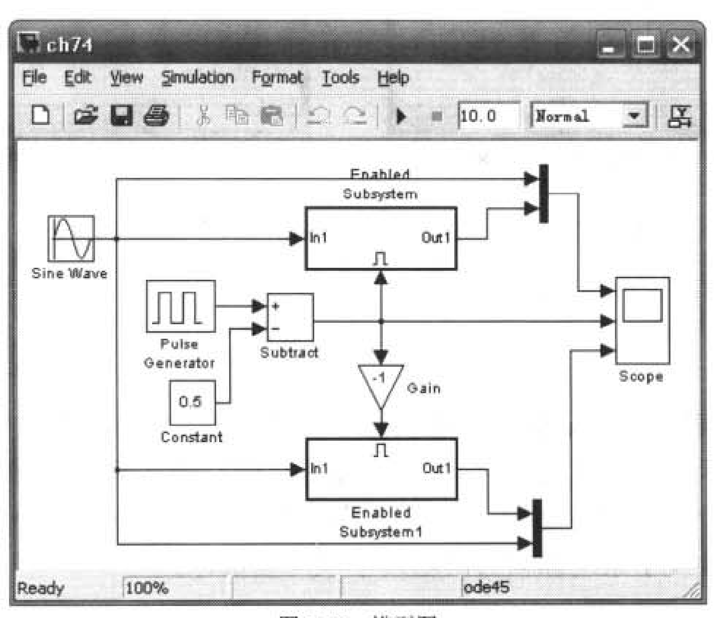


图 7-30　模型图

在本例中使用了两个使能子系统，为了能够更加清晰地了解使能子系统的功能，对同一输入信号取截然相反的输入控制信号，对比子系统的输出。为了构造截然相反的输入控制信号，我们才有了 Gain 模块和 Constant 模块，当然读者可自己采用不同的模块来构造。

1. 进行模块参数设置

（1）Pulse Generator 模块参数设置。其中 Amplitude 为 1，Period 为 1，Pulse Width 为 50，Phase delay 为 0。

（2）Sine Wave 模块参数设置。其中 Amplitude 为 1，Bias 为 0，Frequency 为 1，Phase delay 为 0，Sample time 为 0。

（3）Constant 模块参数设置。其中 Constant value 为 0.5。

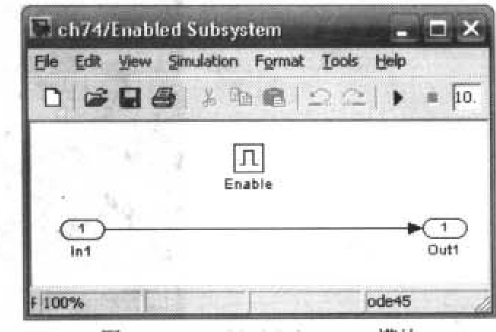


图 7-31 Enabled Subsystem 模块

（4）Gain 模块参数设置。其中 Gain 为-1。

（5）Enabled Subsystem 模块和 Enabled Subsystem1 模块采用同样的设置。双击则弹出如图 7-31 所示的窗口，然后双击 Enable 模块，弹出如图 7-32 所示的对话框。

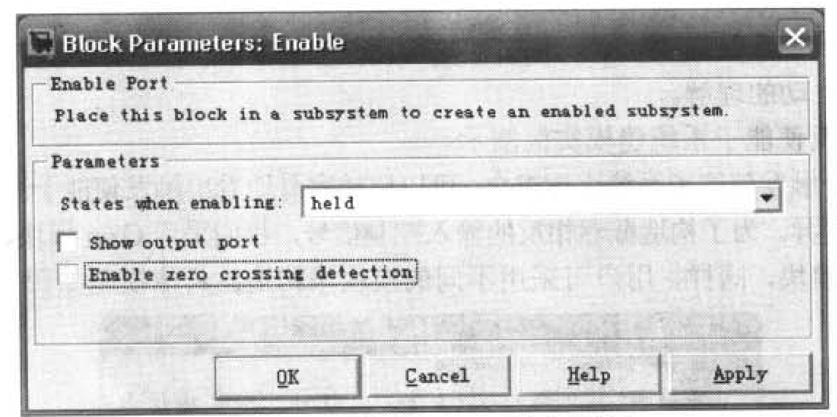


图 7-32 Enabled Subsystem 模块参数设置

2. 运行仿真

模型系统参数采用缺省设置，仿真结果如图 7-33 所示。

3. 结果分析

从图 7-33 所示的结果图中可见，第 1 幅图为 Pulse Generator 模块和 Constant 求和信号直接输入使能模块的结果图和正弦信号图，第 2 幅图为 Pulse Generator 模块信号，第 3 幅图为 Pulse Generator 模块和 Constant 求和信号取反后输入使能模块的结果图和正弦信号图。

从结果图中可以看出：当 Pulse Generator 模块产生的信号为正时，第 1 个使能子系统直接输出正弦信号，而第 2 个使能子系统的信号则保持不变。当 Pulse Generator 模块产生的信号为负时，情况则正好相反，第 2 个使能子系统直接输出正弦信号，而第 1 个使能子系统的信号则保持不变。

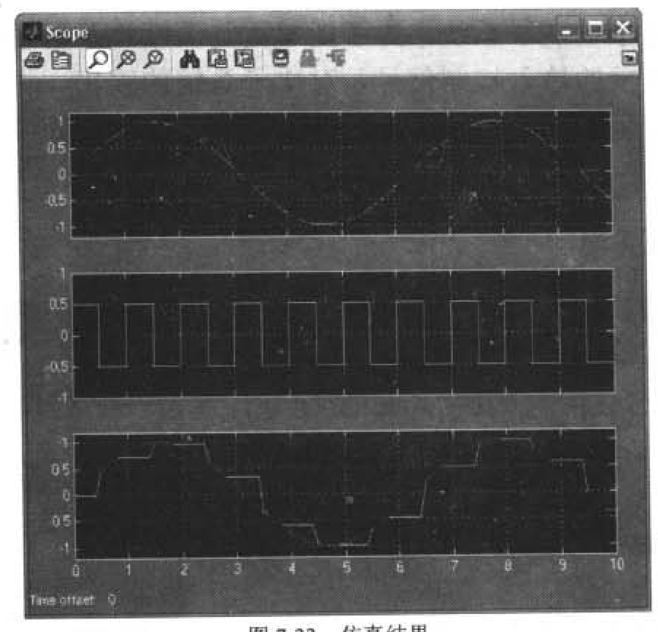


图 7-33　仿真结果

7.3.3　触发使能子系统及其实例

所谓触发使能子系统，就是同时融合了触发子系统和使能子系统的功能。

如同介绍触发子系统模块一样，用实例的形式进行讲解，这样能够让读者对触发使能子系统有更加清晰生动的理解。

【例 7.5】触发使能子系统建模实例演示。

本例采用 4 个触发使能子系统进行组合，可以比较容易地看出触发使能子系统的功能，以及不同设置之间的差异。为了构造截然相反的输入控制信号，我们采用 Gain 模块、Pulse Generator 模块和 Constant 模块，同样，用户可采用不同的模块来构造。具体模型如图 7-34 所示，保存

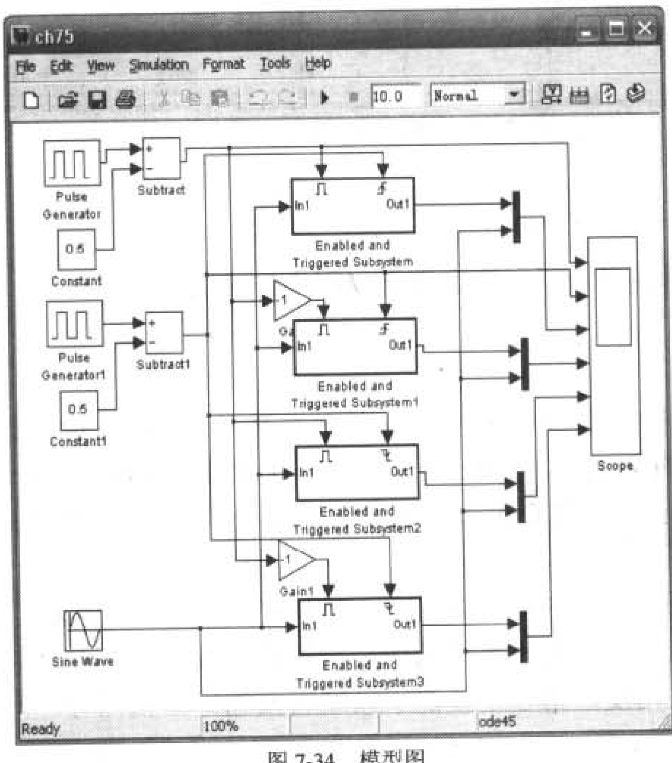


图 7-34　模型图

为 ch75.mdl。

1. 进行模块参数设置

（1）Pulse Generator 模块参数设置。其中 Amplitude 为 1，Period 为 1，Pulse Width 为 50，Phase delay 为 0。

（2）Pulse Generator1 模块参数设置。其中 Amplitude 为 1，Period 为 1，Pulse Width 为 50，Phase delay 为 0。

（3）Sine Wave 模块参数设置。其中 Amplitude 为 1，Bias 为 0，Frequency 为 1，Phase delay 为 0，Sample time 为 0。

（4）Constant 模块和 Constant1 模块参数设置。其中 Constant value 为 0.5。

（5）Gain 和 Gain1 模块参数设置。其中 Gain 为−1。

（6）Enable and Triggered Subsystem 模块和 Enable and Triggered Subsystem1 模块采用同样的设置。双击此模块，则弹出如图 7-35 所示的窗口，接着双击 Triggered 模块，弹出如图 7-36 所示的对话框，在 Triggered Type 下拉列表框中选择 rising 选项。然后双击 Enable 模块，弹出如图 7-37 所示的对话框，在 States when enabling 下拉列表框中选择 held 选项。

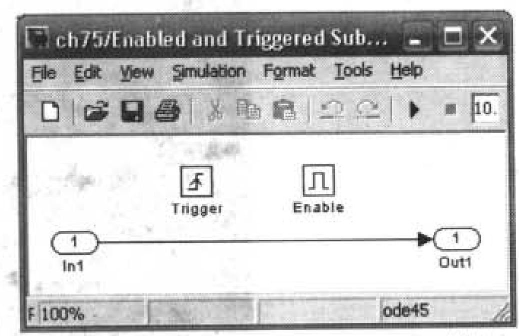


图 7-35 Enable and Triggered Subsystem 模块

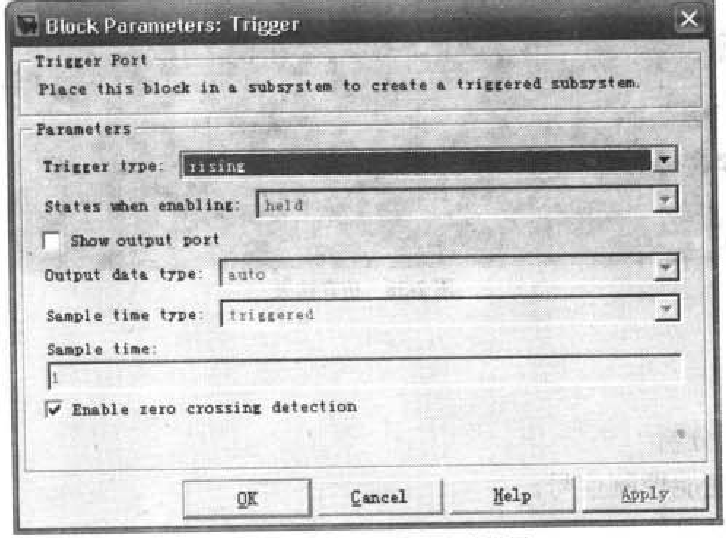


图 7-36 Triggered 模块参数设置

图 7-37 Enable 参数设置

（7）Enable and Triggered Subsystem2 模块和 Enable and Triggered Subsystem3 模块采用同样的设置。与（6）中基本相同，只是在 Triggered Type 下拉列表框中选择 falling 选项。

2. 进行仿真

模型系统参数采用默认值，仿真结果如图 7-38 所示。

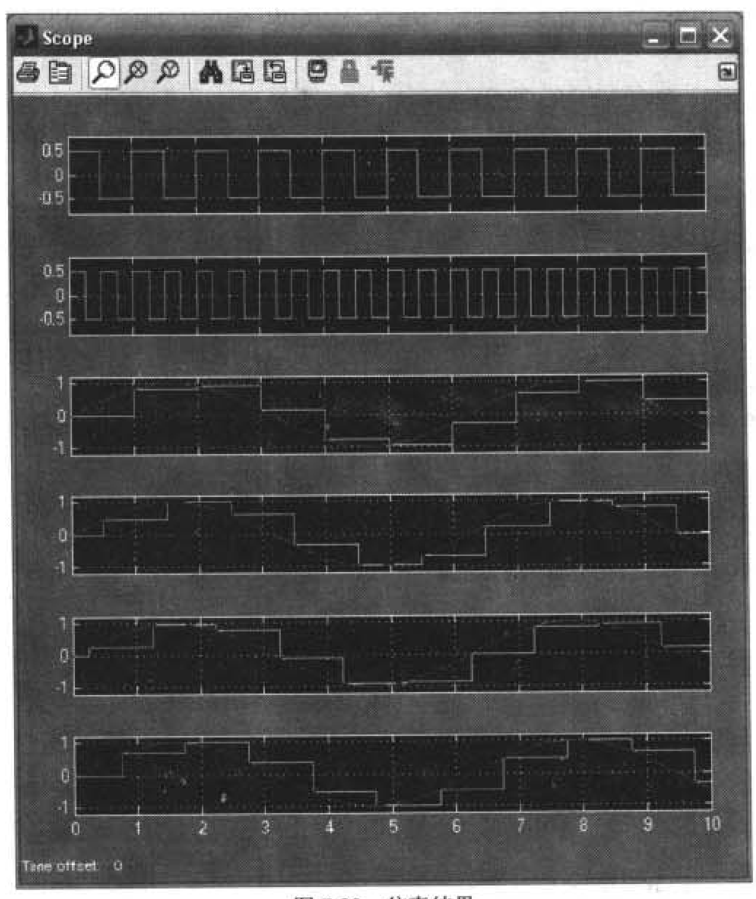


图 7-38 仿真结果

3. 结果分析

（1）子系统设置分析

- 第 1 幅图是使能控制信号。
- 第 2 幅图是触发控制信号。
- 第 3 幅是 Pulse Generator 模块和 Constant 求和信号直接输入使能触发模块，并且触发控制为上升沿触发子系统的结果图和正弦信号图。
- 第 4 幅是 Pulse Generator 模块和 Constant 求和信号取反输入使能触发模块，并且触发控制为上升沿触发子系统的结果图和正弦信号图。

（2）结果分析

- 第 1 幅图表明，在输入使能触发子系统模块的使能信号为正，并且触发信号为上升沿触发信号时，输出才会变化，其他情况都保持常值，这是因为触发模块采用了 held 选项。
- 第 2 幅图表明，在输入使能触发子系统模块的使能信号为负，并且触发信号为上升沿

触发信号时，输出才会变化，其他情况都保持常值，这是因为触发模块采用了 held 选项。

- 第 3 幅图表明，在输入使能触发子系统模块的使能信号为正，并且触发信号为下降沿触发信号时，输出才会变化，其他情况都保持常值，这是因为触发模块采用了 held 选项。
- 第 4 幅图表明，在输入使能触发子系统模块的使能信号为负，并且触发信号为下降沿触发信号时，输出才会变化，其他情况都保持常值，这是因为触发模块采用了 held 选项。

7.3.4 交替执行子系统及其实例

交替执行子系统就是在对输入信号进行判断的基础上，然后选择相应的输出端口，使得相应的交替执行子系统被执行。

下面通过一个实例来进行说明，以更加清晰明了。

【例 7.6】 交替执行子系统建模实例演示。

建立模型如图 7-39 所示，这是一个对 Switch Case 模块输入信号进行端口选择，然后执行相应的 Switch Case Action Subsystem 子系统，输入信号为 1，2，3，4。其中 4 个 step 模块是用来构造 Switch Case 模块的输入信号。

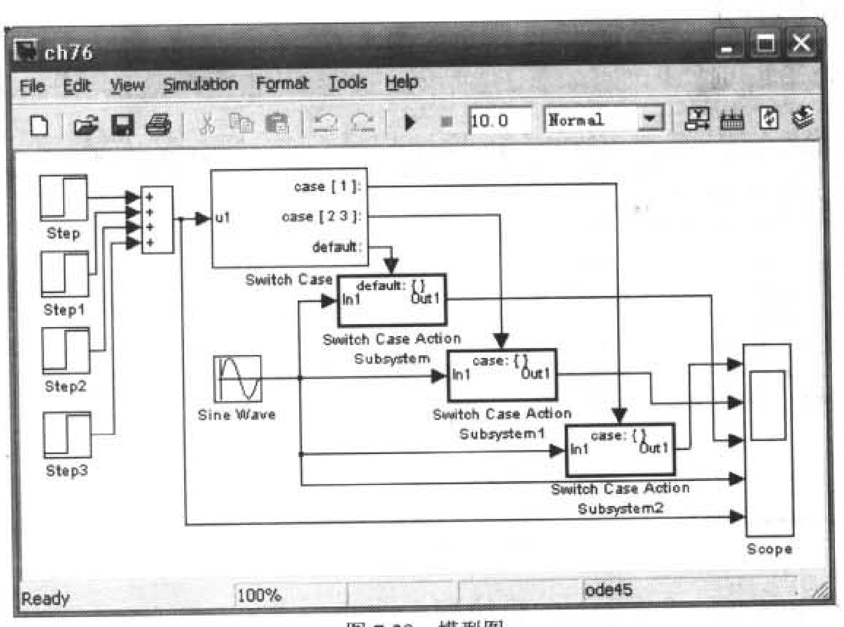


图 7-39 模型图

1. 对模块参数进行设置

（1）Step 模块设置。双击 Step 模块参数对话框，如图 7-40 所示，其中 Step Time 为 0，Initial value 为 0，Final value 为 1，Sample time 为 0。

- Step1 模块参数设置。其中 Step time 为 2，Initial value 为 0，Final value 为 1，Sample time 为 0。
- Step2 模块参数设置。其中 Step time 为 4，Initial value 为 0，Final value 为 1，Sample time 为 0。

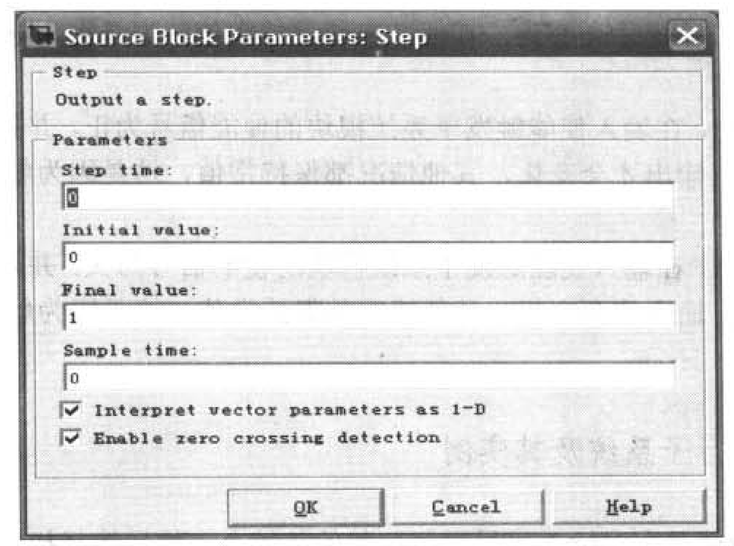


图 7-40　Step 模块设置

- Step3 模块参数设置。其中 Step time 为 6，Initial value 为 0，Final value 为 1，Sample time 为 0。

（2）Sine Wave 模块参数设置。其中 Amplitude 为 1，Bias 为 0，Frequency 为 2，Phase Delay 为 0，Sample time 为 0。

（3）Switch Case 模块参数设置。双击 Switch Case 模块参数对话框，如图 7-41 所示，其中 Case condition 为{1，[2，3]}。

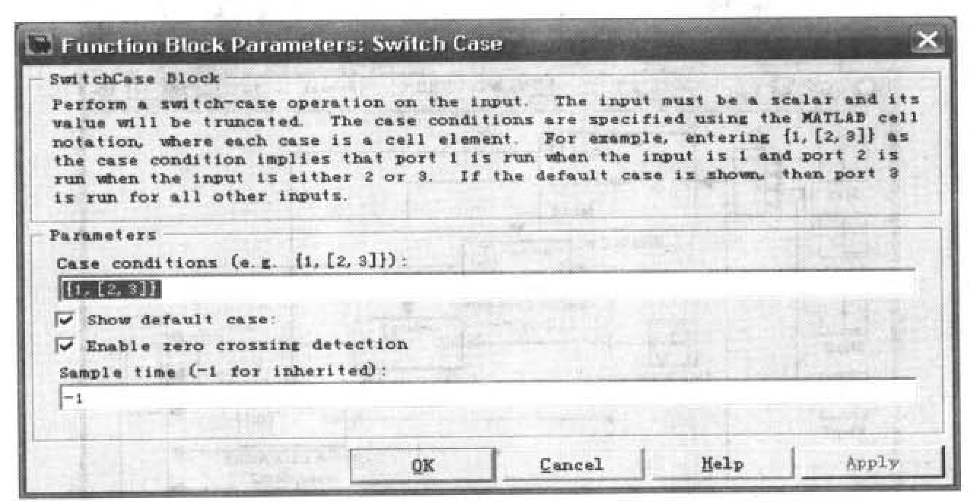


图 7-41　Switch Case 模块设置

（4）Switch Case Action Subsystem 模块和 Switch Case Action Subsystem1/2 模块参数设置。与不同的 Switch Case 模块端口连接，显示会不一样，可对比 Switch Case Action Subsystem 模块和 Switch Case Action Subsystem1 模块。双击 Switch Case Action Subsystem 模块，显示如图 7-42 所示。双击 Switch Case Action Subsystem1/2 模块，如图 7-43 所示。双击图 7-42 中的 Action port 模块，弹出参数对话框，如图 7-44 所示，在此不改变默认参数。

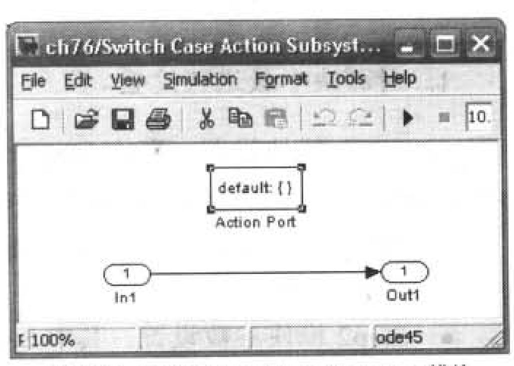


图 7-42　Switch Case Action Subsystem 模块

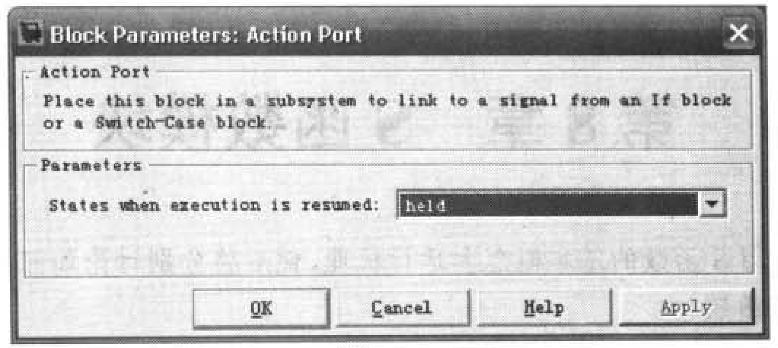


图 7-43　Action port 模块参数设置

2. 运行结果

结果如图 7-44 所示。

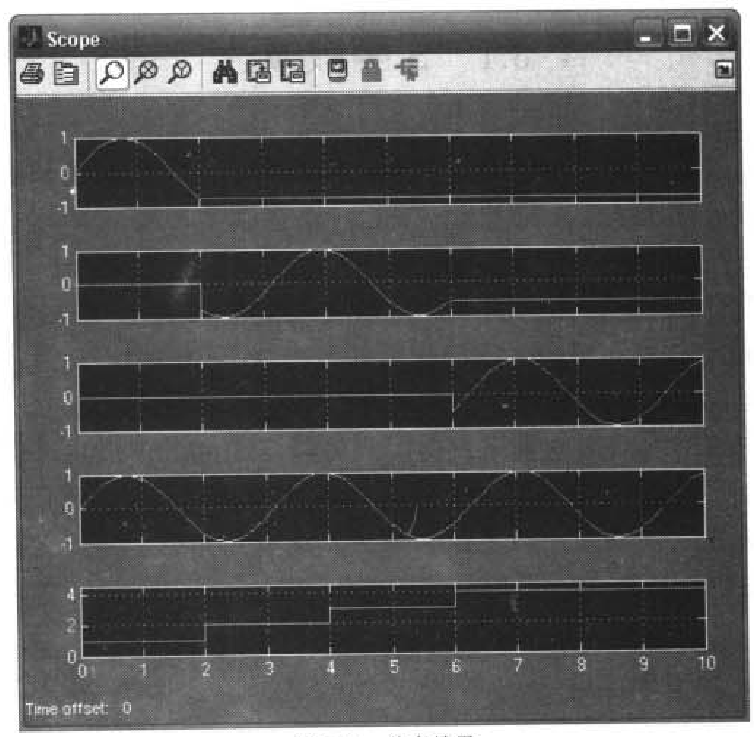


图 7-44　仿真结果

3. 结果分析

- 第 1 幅图示相对于 Switch Case 模块 Case1 端口的结果。
- 第 2 幅图示相对于 Switch Case 模块 Case2 端口的结果。
- 第 3 幅图示相对于 Switch Case 模块 Default 端口的结果。
- 第 4 幅图示是 Sine Wave 模块信号。第 5 幅图是 Switch Case 模块的输入信号。

可以看出，在 Switch Case 模块输入信号为 1 时，这正好对应于 Case1 情况，则 Switch Case Action Subsystem2 模块执行，输出结果为第 1 幅图。当输入信号为 2 或 3 时，Switch Case Action Subsystem1 模块执行，输出结果为第 2 幅图。当 Switch Case 模块输入信号为 4 时，则执行 Default 端口连接的 Switch Case Action Subsystem 模块，输出结果为第 3 幅图。

第 8 章 S 函数模块

本章将首先介绍 S 函数的基本概念和运行机理，然后将分别讨论如何使用 M 文件或其他高级语言来编写 S 函数。

本章主要内容：

- S 函数概述
- 编写 M 文件形式的 S 函数
- 编写 C Mex 文件形式的 S 函数

8.1 S 函数概述

S 函数，即系统函数，在很多情况下都是非常有用的，它是扩展 Simulink 功能的强有力工具。它使用户可以利用 MATLAB、C 语言、C++语言以及 Fortran 等语言的程序创建自定义的 Simulink 模块。例如，对一个工程的几个不同的控制系统进行设计，而此时已经用 M 文件建立了一个动态模型，在这种情况下，可以将模型加入到 S 函数中，然后使用独立的 Simulink 模型来模拟这些控制系统。这样先前的努力就不会白费，而且模型还可以方便地重复使用。S 函数还可以改善仿真的效率，尤其是在带有代数环的模型中。

S 函数使用一种特殊的调用规则来使得用户可以与 Simulink 的内部解法器进行交互，这种交互和 Simulink 内部解法器与内置的模块之间的交互非常相似，而且可以适用于不同性质的系统，例如连续系统、离散系统以及混合系统。这里先介绍一下什么是 S 函数、S 函数的作用和原理、S 函数的相关概念，最后举例说明。

8.1.1 什么是 S 函数

S 函数是对一个动态系统的计算机程序语言描述。S 函数可以使用 MATLAB 或者 C 语言写成。用 C 语言写成的 S 函数需要用 Mex 工具编译成 Mex 文件。与其他的 Mex 文件一样，它们在需要的时候动态地链接到 MATLAB。

S 函数使用一种特殊的调用语法，通过它可以与 ODE 求解器进行交互。这种交互同求解器与 Simulink 内建模块之间的交互非常相似。

S 函数的形式非常全面，它包括连续、离散和混合系统，因此，几乎所有的 Simulink 模型都可以描述为 S 函数。

通过 User-Defined Functions 库中的 S-Function 模块，可以将 S 函数加进 Simulink 模型，使用 S-Function 模块对话框可以指定 S 函数的名字，如图 8-1 所示。

在该例中，模型包含有一个 S-Funcnon 模块，模块引用的源程序的名字是 system，它可以是一个 C Mex 文件或者 M 文件，如果存在具有相同名字的 C Mex 文件和 M 文件，S 函数优先使用 C Mex 文件。

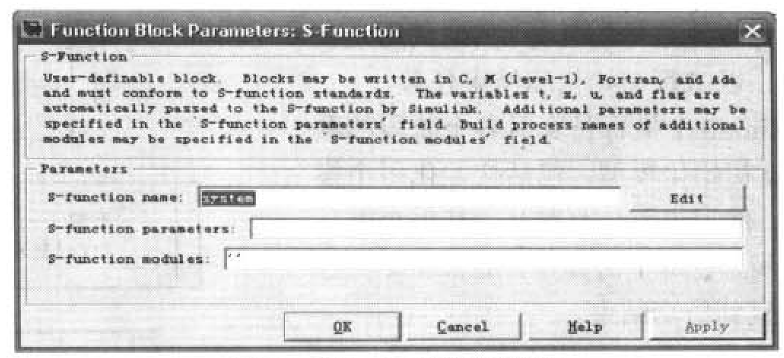


图 8-1 S-Function 模块参数设置

可以使用 Simulink 的模板工具为 S-Function 模块创建一个定制的对话框和图标。模板对话框使得为 S 函数指定附加的参数变得更容易一些。

8.1.2 S 函数的作用与原理

S 函数最通常的用法是创建一个定制的 Simulink 模块，可以在许多应用程序中使用 S 函数，包括：

（1）在 Simulink 中加进新的通用模块；

（2）将已存在的 C 代码合并入一个仿真中；

（3）将一个系统描述为一系列的数学方程；

（4）使用图形动画。

使用 S 函数的一个优点是可以创建一个通用的模块，在模型中可以多次使用它，使用时只需要改变它的参数值即可。

Simulink 模型中的每一个模块都有如下的共同特征；一个输入向量 U，一个输出向量 Y，以及一个状态向量 X，如图 8-2 所示。



图 8-2 Simulink 系统模型

状态向量可能包括连续状态、离散状态或连续状态与离散状态的组合。输入、输出和状态之间的数学关系可以用式（8.1）所示的方程来表达：

$y = f_0(t,x,u)$(输出)

$x_c^{'} = f_d(t,x,u)$(导数)

$x_{d_{k+1}} = f_u(t,x,u)$(更新)

其中，$x = x_c + x_d$ （8.1）

Simulink 将状态向量分为两部分：连续状态和离散状态。连续状态占据着向量的第一部分，离散状态占据第二部分。对于没有状态的模块，x 是一个空的向量。在 Mex 文件 S 函数中，有两个独立的状态向量：一是离散状态向量，另一个是连续状态向量。

1. 仿真阶段和 S 函数程序

在仿真的特定阶段，Simulink 反复调用模型中的每一个模块，以执行诸如计算输出、更

新离散状态或者计算导数这样的任务。其他调用是在仿真的开始或结束，以执行初始化和结束任务。

图 8-3 显示了 Simulink 执行仿真时各个阶段的顺序。首先，Simulink 初始化模型，包括初始化每个模块，也包括 S 函数，然后进入仿真循环，其每个循环经过作为一个仿真步，在每个仿真步，Simulink 执行用户 S 函数模块，直到仿真结束。

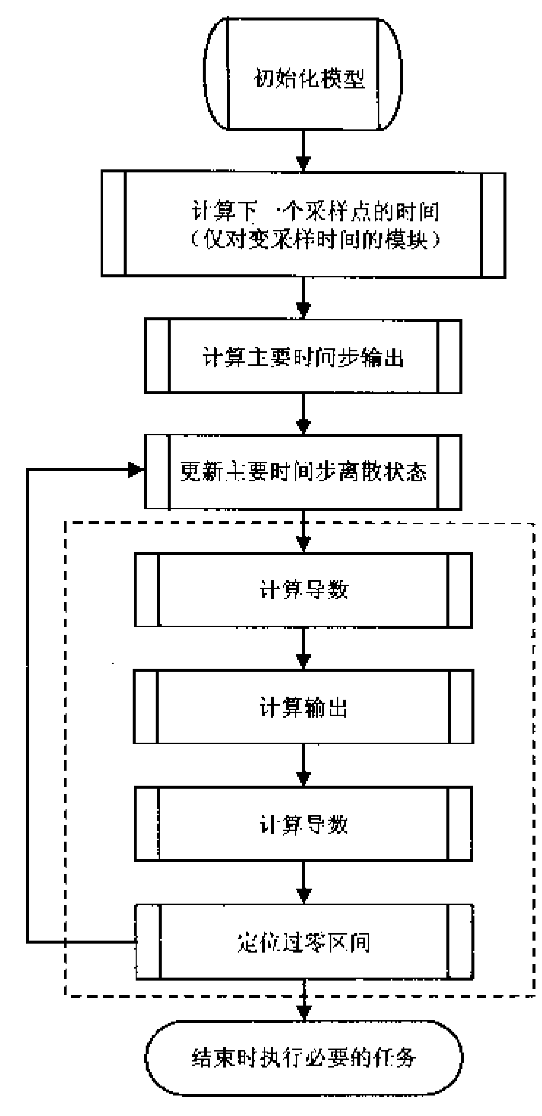


图 8-3 Simulink 执行仿真时各个阶段的顺序

Simulink 在模型中反复地调用 S 函数程序，以执行每一个阶段需要的任务。这些任务包括。

（1）初始化。在第一仿真循环之前，Simulink 初始化 S 函数，此阶段 Simulink 执行以下工作：

- 初始化 SimStructure，SimStructure 是一个包含关于 S 函数信息的 Simulink 结构。
- 设置输入和输出端口的数量和大小。
- 设置模块采样时间。
- 分配存储空间并估计数组大小。

（2）计算下一采样点。如果选择了变步长积分程序，该阶段计算下一变点时间，即计算下一时间步。

（3）计算主要时间步的输出。该调用完成后，模块的所有输出端口对于当前时间步是合法的。

（4）更新主要时间步的离散状态。在该调用中，所有模块应当执行各个时间步一次。

（5）积分。用于具有连续状态或非采样过零区间的模型。如果 S 函数具有连续状态，Simulink 在辅助时间步调用 S 函数的输出和导数。这就是 Simulink 可以计算 S 函数状态的原因。如果 S 函数（仅为 C Mex）具有非采样过零区间，Simulink 将在辅助时间步调用 S 函数输出和过零成分，所以可以定位过零区间。

2. M 文件 S 函数和 C Mex 文件 S 函数

在 M 文件形式的 S 函数中，S 函数程序是作为 M 文件子程序的形式实现的。在 C Mex 文件形式的 S 函数中，它们是以 C 语言函数实现的。S 函数程序的名字和它们执行的功能在 M 文件和在 C Mex 文件 S 函数中是相同的。

对于 M 文件形式的 S 函数，Simulink 传递一个标志参数给 S 函数，该标志表示当前的仿真阶段。必须编写 M 代码调用合适的函数以得到每一个标志的值。对于一个 C Mex 文件形式的 S 函数，Simulink 直接调用 S 函数程序，表 8-1 列出了各个仿真阶段相应的 S 函数程序以及相关的 M 文件形式的 S 函数的标志值。

C Mex 文件形式的 S 函数程序必须具有与表 8-1 列出的完全相同的名字。在 M 文件形式的 S 函数中，必须提供基于标志值的调用合适的 S 函数程序的代码。一个模板 M 文件 S 函数 sfuntmpl.m 位于/toolbox/simulink/blocks 目录中。该模板使用一个 Switch 语句处理标志的值。所有需要做的就是将你的代码放在恰当的 S 函数程序中。

表 8-1　各个仿真阶段相应的 S 函数程序以及相关的 M 文件形式的 S 函数的标志值

仿 真 阶 段	S 函数程序	标志 M 文件形式的 S 函数
初始化	mdlInitializeSizes	flag=0
计算下一个采样点（可选）	mdlGetTimeOfNextVarHit	flag=4
计算输出	mdlOutputs	flag=3
更新离散状态	mdlUpdate	flag=2
计算导数	mdlDerivatives	flag=1
仿真任务结束	mdlTerminate	flag=9

在 C Mex 文件形式的 S 函数中，Simulink 直接调用当前仿真阶段的适当的 S 函数程序。一个用 C 语言写成的样板 S 函数 sfuntmpl_basic.c 或 sfuntmpl_doc.c（这个更详细）位于 Smulink/src 目录中。同一目录下的 sfuctmpl.doc 是一个更为详细的带有注释的模板。

在开发 S 函数时，建议使用 M 文件或 C Mex 文件模板。

8.1.3　S 函数的有关概念

理解直接馈通、动态可变输入、设置采样时间和偏移量等这些关键的概念，将有助于正确地创建 S 函数。

1. 直接馈通

直接馈通的意思是输出或可变采样时间直接由某一端口的输入值控制。通常只要 S 函数满足如下条件它就具有直接馈通：

（1）它的输出函数（mdlOutputs 或 flag==3）是输入的函数，即在 mdlOutputs 函数中，如果使用了输入 u 的等式，就是直接馈通的。输出可能还包括图形化输出，在有 XYGraph 显示器的情况下。

（2）它是一个可变采样时间的 S 函数（调用 mdlGetTimefNextVarhit 或 flag= =4），并且计算下一个采样点需要用到的输入 u。

正确地设置直接馈通非常重要，因为它会影响到模型中各个模块的执行顺序，并且会被用来检测代数循环。

2. 动态可变输入

S 函数可以被编写为支持任意的输入宽度。在这种情况下，当仿真开始后，实际的输入个数是通过计算驱动 S 函数的输入向量的元素的个数来动态地确定的，输入个数也可用来确定连续状态的个数、离散状态的个数和输出的个数。

要指明输入的个数是动态变化的，指定它的 sizes 结构的某些域的值为−1，该结构是调用 mdlInitialize 返回的。当调用 S 函数时，可以用 lenth(u)来确定实际的输入个数。

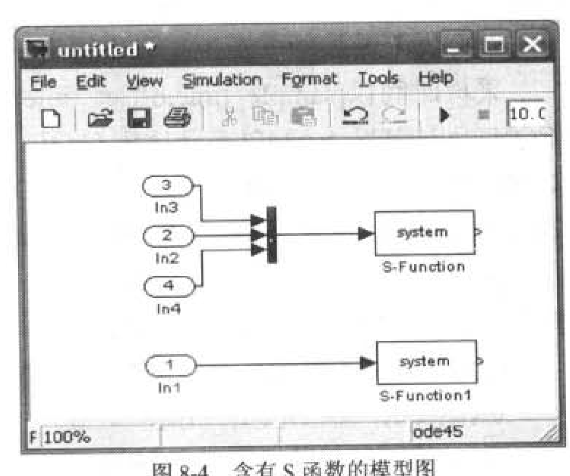


图 8-4　含有 S 函数的模型图

例如，图 8-4 显示了一个模型中的同一个 S 函

数的两个实例。

图 8-4 中上面的一个 S 函数模块由一个具有三个元素的输出向量的模块驱动。下面的一个 S 函数模块由一个具有一个标量输出的模块驱动，通过指定该 S 函数模块具有动态变化的输入个数。可以使同一个 S 函数适用上面两种情况。Simulink 自动地用具有合适的输入个数的向量调用该模块。同样的，如果模块的其他诸如输出个数、离散状态或者连续状态这样的一些特征也可被指定为动态变化的，Simulink 确定这些向量的长度为与输入向量的长度相同。

3. 设置各采样时间点及其偏移值

无论是 M 文件还是 C Mex 文件 S 函数，在指定何时执行 S 函数时，都具有较高的灵活性。Simulink 提供以下采样时间供选择：

（1）连续采样时间：对于具有连续状态和具有非采样过零区间的 S 函数，这类 S 函数的输出在辅助时间步改变。

（2）连续但在辅助时间步采样时间固定：在每个主要仿真步需要执行，但在辅助时间步不改变值的 S 函数。

（3）离散采样时间：如果 S 函数模块的行为是离散时间间隔的一个函数，可以定义采样时间，以控制什么时候 Simulink 调用该模块，也可以定义每一个采样时间点的延迟偏移值。偏移值的大小不能超过相应的采样时间。

一个采样时间点发生的时刻由下面的公式确定：

$$采样时间=(n\times 周期)+偏移值$$

式中，n 是一个整数，它的第一个值是 0。

当采样时间给定后，Simulink 在每一个采样时间点调用 mdlOutput 和 mdlUpdate 程序。采样时间点由上面给出的公式确定。在编写一个单一速率的离散 S 函数时，在输出和更新函数中没有必要明确地检测采样时间点。Simulink 只在适当的采样时间点上调用 S 函数。

（4）变采样时间：采样点间间隔可以变化的离散采样时间。在每个仿真步开始，具有变采样时间的 S 函数为下一采样点时间积分。

（5）继承采样时间：有时候，一个函数模块没有固有的采样时间属性（也就是说，它可能是连续的也可能是离散的，这取决于系统中另外一些模块的采样时间），可以指定这些模块的采样时间为 inherited，一个简单的例子是 Gain 模块从驱动它的模块那儿继承采样时间。模块可以从驱动模块、目标模块、系统中最快采样时间中继承采样时间。要设置一个模块的采样时间是从别处继承过来，将采样时间设为−1。

S 函数可以是单采样率或多采样率，多采样率 S 函数具有多个采样时间。

采样时间以[sample_time,offset_time]这样的格式对来指定。合法的采样时间对有：

[CONTINUOUS_SAMPLE_T1ME,O,O}

[CONTINUOUS_SAMPLE_T1ME,FIXED_IN_MINOR_STEP_OFFSET]

{discret_sample_time_period,offset]　　(需要实数值)

[VARIABLE_SAMPLE_TIME,0,0]

其中，CONTINUOUS_SAMPLE TIME=O.O

FIXED_IN_MINOR_STEP_OFFSET=1.0

VARIABL_SAMPLE_TIME=-2.0

也可以指定采样时间由驱动模块继承，此时，只有一个采样时间对

[INHERITED_SAMPLE_T1ME,0.O]

或

[INHERITED_SAMPLE_TIEM,FIXED_IN_MINOR_STEP_OFFSET]

其中：INHERITED_SAMPLE_TIME=-1.0

指定采样时间时应遵循以下一些规则：

（1）一个在辅助积分步改变的连续 S 函数，指定[CONTINUOUS_SAMPLE_TIME,0.0]采样时间。

（2）一个在辅助积分步不改变的连续 S 函数，应该指定[CONTINUOUS_SAMPLLE_TIME,F1KED_IN_MINOR_STEP_OFFSET]采样时间。

（3）一个指定速率下改变的离散 S 函数，应该指定离散采样时间对：[discrete_sample_time_period,offset]。

其中：discrete_sample_period>0.0,并且，0.0<=offset<discrete_sample_period。

（4）在变化速率下改变的离散 S 函数，可以指定变步长离散采样时间：[VARIABLE_SAMPLE_TIME,0.0]

对于变步长离散任务，调用 mdlGetTimeOfNextvarHit 子程序来获取下一采样点时间。如果 S 函数没有固有的采样时间，就必须继承采样时间。

（5）一个即使在辅助积分步也随其输入改变而改变的 S 函数，指定采样时间

[INHERITED_SAMPLE_TIME,0.0]。

（6）一个随输入改变而改变，但在辅助积分步不改变的 S 函数，就指定[INHERITED_SAMPLE_TIME，FIXED_IN_MINOR_STEP_OFFSET]采样时间。

8.1.4 S 函数的例子

有一些 S 函数的示例保存在 MATLAB 根目录下的以下两个子目录下：simulink/src 目录下保存有 C Mex 文件；toolbox/simulink/blocks 目录下保存有 M 文件。

在 simulink/blocks 目录下包含有一些 M 文件形式的 S 函数，这些文件有：csfunc.m, dsfunc.m,vsfunc.m,mixed.m,vdpm.m,simom.m,simon2.m,limintm.m,sfun_varargm.m,vlimintm.m 和 vdkmintm.m。表 8-2 中列出了上述这些 S 函数及其简要说明。

表 8-2　　M 文件形式的 S 函数示例

文　件　名	说　　明
Csfunc.m	以状态空间形式定义一个连续系统
Dsfunc.m	以状态空间形式定义一个离散系统
Vsfunc.m	示范如何创建变步长模块，该模块实现一个变步长延迟，第一个输入延迟由第二个输入确定的时间
Mixed.m	实现由一个连续积分器和一个单位延迟器串联混合系统
Vdpm.m	实现 Van der pol 等式
Simom.m	一个具有 A、B、C、D 内部矩阵的状态空间 M 文件 S 函数例子，该 S 函数实现：$dx/dt=Ax+By;y=Cx+Du$；其中，x 是状态向量，u 是输入向量，y 是输出向量，A、B、C、D 矩阵嵌入 M 文件
Simom2.m	一个具有 A、B、C、D 外部矩阵的状态空间 M 文件 S 函数例子，状态空间结构与 simom.m 一样，但是 A、B、C、D 矩阵由文件参数外部提供
Limintm.m	实现连续限定积分器，其输出被限制在上下边界，并限制其初始条件
Sfun_varargm.m	这是一个显示如何使用 MATLAB vararg 灵活性的 M 文件 S 函数例子
Vlimintm.m	一个连续限定积分器 S 函数的例子，示范如何使用大小输入为-1 来建立一个提供动态输入/状态宽度的 S 函数
Vdlmintm.m	一个离散限定积分器 S 函数，与 vlimintm.m 一样，只是积分器是离散的

在 simulink/src 目录下还有 C Mex 文件 S 函数。其中多数有 M 文件的 S 函数副本。表 8-3 中列出了这些 C Mex 文件 S 函数。

表 8-3 C Mex 文件形式的 S 函数示例

文 件 名	说 明
Timestwo.c	一个将其输入乘以 2 的基本 C Mex 文件 S 函数
Csfunc.c	定义一个连续系统的 C Mex 文件 S 函数例子
Dsfunc.c	定义一个离散系统的 C Mex 文件 S 函数例子
Dlimiint.c	实现离散时间限定积分器
Vsfunc.c	示范如何创建变步长模块，该模块实现一个变步长延迟，第一个输入延迟由第二个输入确定的时间
Mixed.c	实现由一个连续积分器（1/s）和一个单位延迟（1/z）串联的混合动态系统
Mixedmex.c	实现一个具有单输出和双输入的混合动态系统
Quantize.c	向量化模块的 Mex 文件例子
Resetint.c	复位积分器
Sftable2.c	二维查找表 S 函数形式
Sfun_dynsize.c	如何动态确定 S 函数输出大小的例子
Sfun_errhndl.c	使用 mdlChechParams S 函数子程序如何检查参数的例子
Sfun_fcncall.c	一个配置函数调用子系统的 S 函数例子
sfun_multiport.c	具有多个输入输出端口的 S 函数例子
Sfun_multirate.c	如何指定基于端口采样时间的 S 函数的例子
Sfun_zc_sat.c	执行 abs(u)的具有非采样过零点的 S 函数的例子，S 函数设计为变步长求解器
Sfuc_zc_sat.c	使用过零区间的饱和的例子
Sfunmem.c	一个一积分步延迟和保持“存储”函数
Vdpm.c	实现 Van der pol 等式
Simomex.c	实现一个单输出、双输入状态空间动态系统，其状态方程为：$dx/dt=Ax+By;y=x'+Du$；其中，x 是状态向量，u 是输入向量，y 是输出向量
Stspace.c	实现一组状态空间方程，通过使用 S 函数模块和模板功能，可以将该函数转换为一个新模块，该 Mex 文件例子与内置的状态空间（state-space）模块功能一样，这是一个输入、输出、状态个数依赖工作空间传递参数的例子
Stvctf.c	执行一个连续传递函数，传递函数多项式由输入向量传递，这对离散时间自适应控制应用非常有用
Stvdct.f	实现离散传递函数，其传递函数多项式由输入向量传递，这对离散时间自适应控制应用非常有用
Limintc.m	实现连续限定积分器
Vdlmint.m	实现一个离散时间向量化限定积分器
Vlimint.m	实现一个向量化限定积分器

8.2 编写 M 文件形式的 S 函数

定义 S 函数模块的 M 文件必须提供关于模型的一些信息，Simulink 在仿真的时候需要用到这些信息，在仿真期间，Simulink、ODE 求解器和 M 文件之间互相作用，以执行具体的任务。这些任务包括定义初始状态和模块属性，计算导数、离散状态和输出。

Simulink 提供了一个 M 文件形式的 S 函数模板，它包括定义一些必要函数的语句和一些注释，这些注释有助于编写 S 函数模块中所必需的代码，模板文件 sfuntmpl.m 位于 MATLAB 根目录下的 toolbox/Simulink/blocks 目录下。

M 文件 S 函数通过调用一系列 S 函数子程序来工作，这些子程序是执行任务所必需的 M 代码函数。表 8-4 列出了 M 文件 S 函数可用的 S 函数子程序。

表 8-4　　M 文件 S 函数可用的 S 函数子程序

S 函数子程序	说　明
mdlInitializesizes	定义基本的 S 函数模块特性，包括采样时间、连续和离散状态的初始条件、数组大小
mdlDerivatives	计算连续状态变量导数
mdlUpdate	更新离散状态，采样时间，主要时间步
mdlOutputs	计算 S 函数的输出
mdlGetTiemOfNextVarHit	以绝对时间计算下一采样点的时间，该子程序仅用于在 mdlInitializeSizes 中指定一个可变离散采样时间的时候
mdlTerminate	执行仿真任务必要的结束操作

创造 S 函数可以分为两个独立的任务：一个是初始化模块特性，包括输入、输出个数，连续和离散状态初始条件，采样时间；另一个是将算法放在合适的 S 函数子程序中。

8.2.1　定义 S 函数模块的属性

要让 Simulink 能够识别一个 M 文件形式的 S 函数，必须提供该 S 函数的一些指定信息，这些信息包括输入、输出和状态的个数以及模块的其他一些属性。

要提供这些信息给 Simulink，在 mdlInitializeSizes 的开始调用 simsizes 函数：

sizes=simsizes

这一函数返回一个 Size 结构。表 8-5 列出了 Size 结构的各个域，并且说明了每个域中包含的信息：

表 8-5　　Size 结构的各个域及每个域中包含的信息

域　名	描　述
Sizes.NumContStates	连续状态的个数
Sizes.NumDiscState	离散状态的个数
Sizes.NumOutputs	输出的个数
Sizes.NumInputs	输入的个数
Sizes.DirFeedthrough	直接馈通标志
Sizes.NumSampleTimes	采样时间的个数

在初始化好 size 结构之后，再调用一次 simsizes 函数：

sys=simsizes（sizes）；

这一语句将 Size 结构中的信息传递给 sys，以便将信息保存起来供 Simulink 使用。

8.2.2　M 文件形式的 S 函数的例子

下面给出一个简单的例子，例子中的 S 函数模块接受一个标量信号的输入并将它乘以 2，如图 8-5 所示。

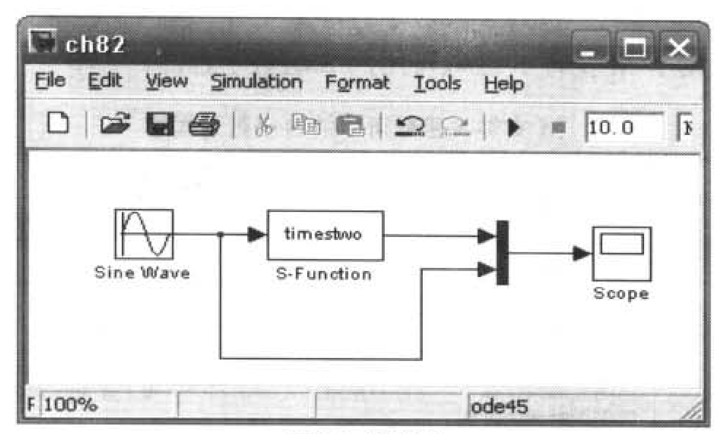


图 8-5 模型图

运行仿真后可以发现，结果与预期的一致，下面我们详细看一下 S-Function 部分是如何实现的。

该 S 函数的 M 文件代码是基于 S 函数模板 sfuntmpl.m 生成的，通过使用该模板，可以创建一个看起来与 C Mex 文件形式的 S 函数非常相似的 M 文件形式的 S 函数。这就使得从 M 文件到 C Mex 文件的转换变得非常容易。

下面是 timestwo.m 模块的 M 文件代码。

```
function [sys,x0,str,ts]=timestwo(t,x,u,flag)
%TIMESTWO S-function whose output is two times its input.
%     This M-file illustrates how to construct an M-file S-function that
%     computes an output value based upon its input.   The output of this
%     S-function is two times the input value:
%
%         y=2 * u;
%
%     See sfuntmpl.m for a general S-function template.
%
%     See also SFUNTMPL.

%     Copyright 1990-2002 The MathWorks, Inc.
%     $Revision: 1.7 $


%
% Dispatch the flag. The switch function controls the calls to
% S-function routines at each simulation stage of the S-function.
%
switch flag,
  %%%%%%%%%%%%%%%%%%%%
  % Initialization %
  %%%%%%%%%%%%%%%%%%%%
```

```
% Initialize the states, sample times, and state ordering strings.
case 0
   [sys,x0,str,ts]=mdlInitializeSizes;

%%%%%%%%%%%%
% Outputs %
%%%%%%%%%%%%
% Return the outputs of the S-function block.
case 3
  sys=mdlOutputs(t,x,u);

%%%%%%%%%%%%%%%%%%%%%%%
% Unhandled flags %
%%%%%%%%%%%%%%%%%%%%%%%
% There are no termination tasks (flag=9) to be handled.
% Also, there are no continuous or discrete states,
% so flags 1,2, and 4 are not used, so return an emptyu
% matrix
case { 1, 2, 4, 9 }
  sys=[];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Unexpected flags (error handling)%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Return an error message for unhandled flag values.
otherwise
  error(['Unhandled flag=',num2str(flag)]);

end

% end timestwo

%
%=============================================================================
% mdlInitializeSizes
%R%eturn the sizes, initial conditions, and sample times for the %S-function.
%=============================================================================
%
function [sys,x0,str,ts]=mdlInitializeSizes()
```

```
sizes=simsizes;
sizes.NumContStates =0;
sizes.NumDiscStates =0;
sizes.NumOutputs     =-1;  % dynamically sized
sizes.NumInputs      =-1;  % dynamically sized
sizes.DirFeedthrough=1;   % has direct feedthrough
sizes.NumSampleTimes=1;

sys=simsizes(sizes);
str=[];
x0 =[];
ts =[-1 0];    % inherited sample time

% end mdlInitializeSizes

%
%=============================================================================
% mdlOutputs
% Return the output vector for the S-function
%=============================================================================
%
function sys=mdlOutputs(t,x,u)
%以下是实现幅度乘以 2 的代码
sys=u * 2;

% end mdlOutputs
```

Simulink 传给 S 函数的前四个输入参数必须是变量 t，x，u 和 flag。

t：时间；

x：状态向量（即使没有状态变量，也必须提供）；

u：输入向量；

flag：控制在每一个仿真阶段调用哪个 S 函数子程序的参数。

Simulink 还要求输出参数 sys，x0，str 和 ts，按给定的顺序给出。这些参数分别是：

sys：是一个一般的返回参数，返回的值取决于 flag 的值。例如，如果 flag=3，sys 包含 S 函数的输出。

x0：初始状态值（如果系统中没有状态，将是一个空的向量）。

str：只是为了与 S 函数模块图的 API 一致而提供的，对于 M 文件形式的 S 函数，将它的值设为一个空矩阵。

ts：一个包含有两列的矩阵，分别是与模块相关联的状态的采样时间及其偏移值；采样时间必须按递增的顺序声明，连续系统的采样时间设为 0。

上面讨论的简单例子中没有状态变量。大多数 S 函数模块需要对状态进行处理，不管它是连续的还是离散的，下面讨论系统的四种普通类型：连续、离散、混合和变步长，都可以在 Simulink 中用 S 函数对它进行建模。

下面所有的例子都是基于 sfuntmpl.m 中的 M 文件形式的 S 函数模板。

1．连续状态的 S 函数的例子

Simulink 包含一个名为 csfunc.m 的函数，它是一个用 S 函数建模的连续状态系统的例子。下面是该 M 文件形式的 S 函数的代码：

```
function [sys,x0,str,ts]=csfunc(t,x,u,flag)
%CSFUNC An example M-file S-function for defining a continuous system.
%    Example M-file S-function implementing continuous equations:
%        x'=Ax+Bu
%        y =Cx+Du
%
%    See sfuntmpl.m for a general S-function template.
%
%    See also SFUNTMPL.

%    Copyright 1990-2002 The MathWorks, Inc.
%    $Revision: 1.9 $


A=[-0.09    -0.01
    1          0];

B=[ 1    -7
    0    -2];

C=[ 0    2
    1   -5];

D=[-3    0
    1    0];

switch flag,

  %%%%%%%%%%%%%%%%%%%%
  % Initialization %
  %%%%%%%%%%%%%%%%%%%%
  case 0,
```

```
    [sys,x0,str,ts]=mdlInitializeSizes(A,B,C,D);

  %%%%%%%%%%%%%%%%%%
  % Derivatives %
  %%%%%%%%%%%%%%%%%%
  case 1,
    sys=mdlDerivatives(t,x,u,A,B,C,D);

  %%%%%%%%%%%%%%
  % Outputs %
  %%%%%%%%%%%%%%
  case 3,
    sys=mdlOutputs(t,x,u,A,B,C,D);

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%
  % Unhandled flags %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%
  case { 2, 4, 9 },
    sys=[];

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  % Unexpected flags %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  otherwise
    error(['Unhandled flag=',num2str(flag)]);

end
% end csfunc

%
%=============================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=============================================================================
%
function [sys,x0,str,ts]=mdlInitializeSizes(A,B,C,D)

sizes=simsizes;
sizes.NumContStates =2;
sizes.NumDiscStates =0;
```

```
sizes.NumOutputs     =2;
sizes.NumInputs      =2;
sizes.DirFeedthrough=1;
sizes.NumSampleTimes=1;

sys=simsizes(sizes);
x0 =zeros(2,1);
str=[];
ts =[0 0];

% end mdlInitializeSizes
%
%==============================================================
% mdlDerivatives
% Return the derivatives for the continuous states.
%==============================================================
%
function sys=mdlDerivatives(t,x,u,A,B,C,D)
%此处求导数
sys=A*x+B*u;

% end mdlDerivatives
%
%==============================================================
% mdlOutputs
% Return the block outputs.
%==============================================================
%
function sys=mdlOutputs(t,x,u,A,B,C,D)
%系统输出
sys=C*x+D*u;

% end mdlOutputs
```

与上例不同的是，当 flag=1 时，该例调用 mdlDerivative 来计算连续状态变量的导数，该系统的状态方程如下：

$$\begin{cases} x' = Ax + Bu \\ y = Cx + Du \end{cases}$$

因此，一般的一系列连续微分方程都可用 csfunc.m 进行建模，它与内建的 State-Space 模块比较相似，在对一个系数为时变的状态空间系统进行建模时，该 S 函数可以作为模板。

2. 离散状态的 S 函数的例子

Simulink 包含一个名为 dsfunc.m 的函数，它是一个用 S 函数建模的离散状态系统的例子。该函数与连续 S 函数的例子 csfunc.m 比较相似，唯一不同的是该例调用的是 mdlUpdate 而不是 mdlDerivative。当 flag=2 时，mdlUpdate 更新状态。对于一个单一采样率的离散 S 函数，Simulink 只在采样点上调用程序 mdlUpdate、mdlOutput 和 mdlGetTimeOfnextVarHit（如果需要）。下面是 M 文件形式 S 函数的代码：

```
function [sys,x0,str,ts]=dsfunc(t,x,u,flag)
%DSFUNC An example M-file S-function for defining a discrete system.
%    Example M-file S-function implementing discrete equations:
%        x(n+1)=Ax(n)+Bu(n)
%        y(n)   =Cx(n)+Du(n)
%
%    See sfuntmpl.m for a general S-function template.
%
%    See also SFUNTMPL.

%    Copyright 1990-2002 The MathWorks, Inc.
%    $Revision: 1.18 $

% Generate a discrete linear system:
A=[-1.3839    -0.5097
    1.0000          0];

B=[-2.5559          0
          0    4.2382];

C=[       0    2.0761
          0    7.7891];

D=[   -0.8141    -2.9334
       1.2426          0];

switch flag,

  %%%%%%%%%%%%%%%%%%%%%
  % Initialization %
  %%%%%%%%%%%%%%%%%%%%%
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes(A,B,C,D);
```

```
%%%%%%%%%%%
% Update %
%%%%%%%%%%%
case 2,
   sys=mdlUpdate(t,x,u,A,B,C,D);

%%%%%%%%%%%
% Output %
%%%%%%%%%%%
case 3,
   sys=mdlOutputs(t,x,u,A,C,D);

%%%%%%%%%%%%%%
% Terminate %
%%%%%%%%%%%%%%
case 9,
   sys=[]; % do nothing

%%%%%%%%%%%%%%%%%%%%%%%%
% Unexpected flags %
%%%%%%%%%%%%%%%%%%%%%%%%
otherwise
   error(['unhandled flag=',num2str(flag)]);
end

%end dsfunc

%
%===========================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the  %%S-function.
%===========================================================================
%
function [sys,x0,str,ts]=mdlInitializeSizes(A,B,C,D)

sizes=simsizes;
sizes.NumContStates =0;
sizes.NumDiscStates =size(A,1);
sizes.NumOutputs      =size(D,1);
```

```
sizes.NumInputs       =size(D,2);
sizes.DirFeedthrough=1;
sizes.NumSampleTimes=1;

sys=simsizes(sizes);

x0 =ones(sizes.NumDiscStates,1);
str=[];
ts =[1 0];

% end mdlInitializeSizes

%
%==============================================================
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
%==============================================================
%
function sys=mdlUpdate(t,x,u,A,B,C,D)

sys=A*x+B*u;

%end mdlUpdate

%
%==============================================================
% mdlOutputs
% Return the output vector for the S-function
%==============================================================
%
function sys=mdlOutputs(t,x,u,A,C,D)

sys=C*x+D*u;

%end mdlOutputs
```

上面的例子符合本章前面讨论的仿真阶段，系统的离散状态方程如下：

$$\begin{cases} x(n+1) = Ax(n) + Bu(n) \\ y(n) = Cx(n) + Du(n) \end{cases}$$

因此，最普通的一系列差分方程都可用 dsfun.m 进行建模。它与内建的 Discrete State-Space 模块比较相似，在给系数为时变的离散状态空间系统建模时，可用 dsfunc.m 作为模板。

3. 混合系统 S 函数的例子

Simulink 包括一个名为 mixed.m 的函数，它是一个用 S 函数进行建模的混合系统的例子（连续状态与离散状态的组合）。处理混合系统非常简单，参数 flag 使系统的连续部分和离散部分分别调用适当的 S 函数子程序，混合 S 函数（或任何多采样率 S 函数）的一个奥妙是 Simulink 在所有的采样时间都调用程序 mdlUpdate、mdlOutput 和 mdlGetTimeOfNextVarHit，这就意味着在这些程序中必须确定正在处理哪一个采样点，并且只对相应的采样点执行更新。

在 mixed.m 的模型中，在一个连续 Integrator 后跟一个离散 Unit Delay，用 Simulink 的模块图表示，如图 8-6 所示。

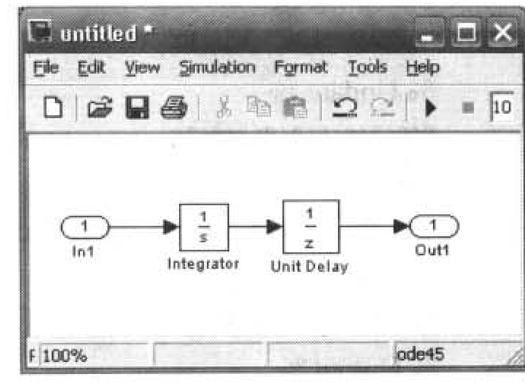


图 8-6 模型图

下面是该 M 文件形式的 S 函数的代码：

```
function [sys,x0,str,ts]=mixedm(t,x,u,flag)
%MIXEDM An example integrator followed by unit delay M-file S-function
%     Example M-file S-function implementing a hybrid system consisting
%     of a continuous integrator (1/s) in series with a unit delay (1/z).
%
%     See sfuntmpl.m for a general S-function template.
%
%     See also SFUNTMPL.

%     Copyright 1990-2002 The MathWorks, Inc.
%     $Revision: 1.28 $


% Sampling period and offset for unit delay.
dperiod=1;
doffset=0;

switch flag

  %%%%%%%%%%%%%%%%%%%%%%
  % Initialization %
  %%%%%%%%%%%%%%%%%%%%%%
  case 0
    [sys,x0,str,ts]=mdlInitializeSizes(dperiod,doffset);
```

```
%%%%%%%%%%%%%%%%
% Derivatives %
%%%%%%%%%%%%%%%%
case 1
  sys=mdlDerivatives(t,x,u);

%%%%%%%%%%%
% Update %
%%%%%%%%%%%
case 2,
  sys=mdlUpdate(t,x,u,dperiod,doffset);

%%%%%%%%%%%
% Output %
%%%%%%%%%%%
case 3
  sys=mdlOutputs(t,x,u,doffset,dperiod);

%%%%%%%%%%%%%%
% Terminate %
%%%%%%%%%%%%%%
case 9
  sys=[];          % do nothing

otherwise
  error(['unhandled flag=',num2str(flag)]);

end

% end mixedm

%
%==============================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%==============================================================
%
function [sys,x0,str,ts]=mdlInitializeSizes(dperiod,doffset)
```

```
sizes=simsizes;
sizes.NumContStates =1;
sizes.NumDiscStates =1;
sizes.NumOutputs      =1;
sizes.NumInputs       =1;
sizes.DirFeedthrough=0;
sizes.NumSampleTimes=2;

sys=simsizes(sizes);
x0 =ones(2,1);
str=[];
ts =[0 0;                    % sample time
     dperiod doffset];

% end mdlInitializeSizes

%
%=============================================================================
% mdlDerivatives
% Compute derivatives for continuous states.
%=============================================================================
%
function sys=mdlDerivatives(t,x,u)

sys=u;

% end mdlDerivatives

%
%=============================================================================
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
%=============================================================================
%
function sys=mdlUpdate(t,x,u,dperiod,doffset)

% next discrete state is output of the integrator
if abs(round((t - doffset)/dperiod) - (t - doffset)/dperiod) < 1e-8
  sys=x(1);
```

```
else
  sys=[];
end

% end mdlUpdate

%
%=============================================================
% mdlOutputs
% Return the output vector for the S-function
%=============================================================
%
function sys=mdlOutputs(t,x,u,doffset,dperiod)

% Return output of the unit delay if we have a
% sample hit within a tolerance of 1e-8. If we
% don't have a sample hit then return [] indicating
% that the output shouldn't change.
if abs(round((t - doffset)/dperiod) - (t - doffset)/dperiod) < 1e-8
 sys=x(2);
else
 sys=[];
end

% end mdlOutputs
```

4. 变步长 S 函数的例子

名为 vsfunc.m 的 M 文件是使用变步长的一个 S 函数的例子，该例当 flag=4 时调用 mdlGetTimeOfNextvarHit，因为计算下一个采样时间取决于输入 u，所以该模块具有直接馈通。通常，使用输入计算下一个采样时间（flag=4）的所有模块都需要直接馈通。下面是 M 文件形式的 S 函数的代码：

```
function [sys,x0,str,ts]=vsfunc(t,x,u,flag)
%VSFUNC Variable step S-function example.
%     This example S-function illustrates how to create a variable step
%     block in Simulink.   This block implements a variable step delay
%     in which the first input is delayed by an amount of time determined
%     by the second input:
%
%        dt       =u(2)
%        y(t+dt)=u(t)
```

```
%
%   See also SFUNTMPL, CSFUNC, DSFUNC.

%   Copyright 1990-2002 The MathWorks, Inc.
%   $Revision: 1.10 $

%
% The following outlines the general structure of an S-function.
%
switch flag,

  %%%%%%%%%%%%%%%%%%
  % Initialization %
  %%%%%%%%%%%%%%%%%%
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;

  %%%%%%%%%%
  % Update %
  %%%%%%%%%%
  case 2,
    sys=mdlUpdate(t,x,u);

  %%%%%%%%%%%
  % Outputs %
  %%%%%%%%%%%
  case 3,
    sys=mdlOutputs(t,x,u);

  %%%%%%%%%%%%%%%%%%%%%%%
  % GetTimeOfNextVarHit %
  %%%%%%%%%%%%%%%%%%%%%%%
  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);

  %%%%%%%%%%%%%
  % Terminate %
  %%%%%%%%%%%%%
  case 9,
    sys=mdlTerminate(t,x,u);
```

```
  %%%%%%%%%%%%%%%%%%%%%
  % Unhandled flags %
  %%%%%%%%%%%%%%%%%%%%%
  case 1,
    sys=[];

  %%%%%%%%%%%%%%%%%%%%%%
  % Unexpected flags %
  %%%%%%%%%%%%%%%%%%%%%%
  otherwise
    error(['Unhandled flag=',num2str(flag)]);

end

% end sfuntmpl

%
%=============================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=============================================================================
%
function [sys,x0,str,ts]=mdlInitializeSizes

%
% call simsizes for a sizes structure, fill it in and convert it to a
% sizes array
%
sizes=simsizes;

sizes.NumContStates =0;
sizes.NumDiscStates =1;
sizes.NumOutputs     =1;
sizes.NumInputs      =2;
sizes.DirFeedthrough=1;
sizes.NumSampleTimes=1;       % at least one sample time is needed

sys=simsizes(sizes);
```

```
%
% initialize the initial conditions
%
x0 =[0];

%
% str is always an empty matrix
%
str=[];

%
% initialize the array of sample times
%
ts =[-2 0];          % variable sample time

% end mdlInitializeSizes

%
%=============================================================================
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
%=============================================================================
%
function sys=mdlUpdate(t,x,u)

sys=u(1);

% end mdlUpdate

%
%=============================================================================
% mdlOutputs
% Return the block outputs.
%=============================================================================
%
function sys=mdlOutputs(t,x,u)

sys=x(1);
```

```
% end mdlOutputs

%
%=============================================================================
% mdlGetTimeOfNextVarHit
% Return the time of the next hit for this block.  Note that the result is
% absolute time.
%=============================================================================
%
function sys=mdlGetTimeOfNextVarHit(t,x,u)

sys=t+u(2);

% end mdlGetTimeOfNextVarHit

%
%=============================================================================
% mdlTerminate
% Perform any end of simulation tasks.
%=============================================================================
%
function sys=mdlTerminate(t,x,u)

sys=[];
% end mdlTerminate
```

子程序 mdlGetTimeOfNextVarHit 返回“下一个采样点的时间”，它是仿真过程中 vsfunc 下一次调用的时间，这就意味着直到下一采样时间点之前该 S 函数不会有输出，在 Vsfunc 中，下一个时间点被设成 t+u(2)，它意味着用第二个输入 u（2）来设置下一次调用 vsfunc 的时间。

5. 传递附加参数

Simulink 经常传递 t，x，u 和 flag 给 S 函数。传递附加参数给 M 文件形式的 S 函数是有可能的。limintm.m 就是这样的一个例子，其代码如下：

```
function [sys,x0,str,ts]=limintm(t,x,u,flag,lb,ub,xi)
%LIMINTM Limited integrator implementation.
%    Example M-file S-function implementing a continuous limited integrator
%    where the output is bounded by lower bound (LB) and upper bound (UB)
%    with initial conditions (XI).
%
%    See sfuntmpl.m for a general S-function template.
```

```
%
%    See also SFUNTMPL.

%    Copyright 1990-2002 The MathWorks, Inc.
%    $Revision: 1.17 $

switch flag

  %%%%%%%%%%%%%%%%%%
  % Initialization %
  %%%%%%%%%%%%%%%%%%
  case 0
    [sys,x0,str,ts]=mdlInitializeSizes(lb,ub,xi);

  %%%%%%%%%%%%%%%
  % Derivatives %
  %%%%%%%%%%%%%%%
  case 1
    sys=mdlDerivatives(t,x,u,lb,ub);

  %%%%%%%%%%%%%%%%%%%%%%%%%
  % Update and Terminate %
  %%%%%%%%%%%%%%%%%%%%%%%%%
  case {2,9}
    sys=[]; % do nothing

  %%%%%%%%%%
  % Output %
  %%%%%%%%%%
  case 3
    sys=mdlOutputs(t,x,u);

  otherwise
    error(['unhandled flag=',num2str(flag)]);
end

% end limintm

%
%=============================================================================
```

```
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=============================================================================
%
function [sys,x0,str,ts]=mdlInitializeSizes(lb,ub,xi)

sizes=simsizes;
sizes.NumContStates =1;
sizes.NumDiscStates =0;
sizes.NumOutputs      =1;
sizes.NumInputs       =1;
sizes.DirFeedthrough=0;
sizes.NumSampleTimes=1;

sys=simsizes(sizes);
str=[];
x0 =xi;
ts =[0 0];     % sample time: [period, offset]

% end mdlInitializeSizes

%
%=============================================================================
% mdlDerivatives
% Compute derivatives for continuous states.
%=============================================================================
%
function sys=mdlDerivatives(t,x,u,lb,ub)

if (x <= lb & u < 0)   | (x>= ub & u>0 )
   sys=0;
else
   sys=u;
end

% end mdlDerivatives

%
%=============================================================================
% mdlOutputs
```

```
% Return the output vector for the S-function
%=============================================================================
%
function sys=mdlOutputs(t,x,u)

sys=x;

% end mdlOutputs
```

8.3 编写 C Mex 文件形式的 S 函数

8.3.1 C Mex 文件形式的 S 函数基本内容

一个定义 S 函数模块的 C Mex 文件也必须提供模型的有关信息，Simulink 在仿真的时候需要用到这些信息。在仿真期间，Simulink、ODE 求解器和 Mex 文件之间互相作用，以执行具体的任务。这些任务包括定义初始状态和模块属性，计算导数、离散状态和各输出值。C Mex 文件形式的 S 函数有着与 M 文件形式的 S 函数相同的结构并且执行相同的功能，Simulink 包含一个编写 C Mex 文件形式的 S 函数的模板，名为 sfuntmpl_basic.c 或 sfuntmpl_doc.c（这个更详细）。

C Mex 文件形式的 S 函数的一般格式如下：

```
/*
 * 必须定义文件名
 */
#define S_FUNCTION_NAME   your_sfunction_name_here
#define S_FUNCTION_LEVEL 2
/*可能要包含的头文件
 *matlabroot/extern/include/tmwtypes.h - General types, e.g. real_T
 *    matlabroot/extern/include/mex.h - MATLAB MEX file API routines
 *matlabroot/extern/include/matrix.h- MATLAB MEX file API routines
 *
 *matlabroot/extern/include/tmwtypes.h - General types, e.g. real_T
 *matlabroot/rtw/c/libsrc/rt_matrx.h- Macros for MATLAB API routines *
 */
#include "simstruc.h"

/*错误处理技术详见 sfuntmpl_doc.c
/*=====================================================================
 * 参数处理，RTW 不支持 *=================================================
```

```
#define MDL_CHECK_PARAMETERS     /* Change to #undef to remove function */
#if defined(MDL_CHECK_PARAMETERS) && defined(MATLAB_MEX_FILE)
  static void mdlCheckParameters(SimStruct *S)
  {
  }
#endif /* MDL_CHECK_PARAMETERS */

#define MDL_PROCESS_PARAMETERS     /* Change to #undef to remove function */
#if defined(MDL_PROCESS_PARAMETERS) && defined(MATLAB_MEX_FILE)
  /* Function: mdlProcessParameters ——————————————————————
*参数处理
   */
  static void mdlProcessParameters(SimStruct *S)
  {
  }
#endif /* MDL_PROCESS_PARAMETERS */

/*=====================================*
 * 配置和执行模块 *
 *=====================================*/
//初始化模块
static void mdlInitializeSizes(SimStruct *S)
{
    int_T nInputPorts =1;   /* number of input ports    */
    int_T nOutputPorts=1;   /* number of output ports */
    int_T needsInput   =1;  /* direct feed through      */

    int_T inputPortIdx =0;
    int_T outputPortIdx=0;


    ssSetNumSFcnParams(S, 0);   /* Number of expected parameters */
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        //此处参数个数不等，需要处理错误
        return;
    }

    /*
     * Configure tunability of parameters.  By default, all parameters are
     * tunable (changeable) during simulation.  If there are parameters that
```

```
 * cannot change during simulation, such as any parameters that would change
 * the number of ports on the block, the sample time of the block, or the
 * data type of a signal, mark these as non-tunable using a call like this:
 *       ssSetSFcnParamTunable(S, 0, 0);
 * which sets parameter 0 to be non-tunable (0).
 */

/* 注册连续和离散系统的状态数 */

ssSetNumContStates(     S, 0); // number of continuous states
ssSetNumDiscStates(     S, 0); // number of discrete states
/*
 * 配置输入端口的个数
 */
if (!ssSetNumInputPorts(S, nInputPorts)) return;
/*
 *设置输入端口的维数，基 0 计数
 */
if(!ssSetInputPortDimensionInfo(S, inputPortIdx, DYNAMIC_DIMENSION)) return;
/*
 * 设置馈通回路
 */
ssSetInputPortDirectFeedThrough(S, inputPortIdx, needsInput);
/*
 * 设置输出端口，设置端口个数
 */
if (!ssSetNumOutputPorts(S, nOutputPorts)) return;
/*
 *设置输出维数，基 0 计数
 */
if(!ssSetOutputPortDimensionInfo(S,outputPortIdx,DYNAMIC_DIMENSION)) return;
/*
 *设置采样时间
 */
ssSetNumSampleTimes(    S, 1);    /* number of sample times                */
/*
 * 设置工作向量维数
 */
ssSetNumRWork(          S, 0);    /* number of real work vector elements    */
ssSetNumIWork(          S, 0);    /* number of integer work vector elements*/
```

```
ssSetNumPWork(          S, 0);   /* number of pointer work vector elements*/
ssSetNumModes(          S, 0);   /* number of mode work vector elements    */
ssSetNumNonsampledZCs( S, 0);   /* number of nonsampled zero crossings    */
/*
 *设置选项
 */
ssSetOptions(           S, 0);   /* general options (SS_OPTION_xx)          */
} /* end mdlInitializeSizes */

#define MDL_SET_INPUT_PORT_FRAME_DATA  /* Change to #undef to remove function */
#if defined(MDL_SET_INPUT_PORT_FRAME_DATA) && defined(MATLAB_MEX_FILE)
  static void mdlSetInputPortFrameData(SimStruct *S,
                                       int       portIndex,
                                       Frame_T   frameData)
  {
  } /* end mdlSetInputPortFrameData */
#endif /* MDL_SET_INPUT_PORT_FRAME_DATA */

#define MDL_SET_INPUT_PORT_WIDTH    /* Change to #undef to remove function */
#if defined(MDL_SET_INPUT_PORT_WIDTH) && defined(MATLAB_MEX_FILE)
  /* Function: mdlSetInputPortWidth
   * Abstract:
   *     输入端口个数动态变化时调用
   */
  static void mdlSetInputPortWidth(SimStruct *S, int portIndex, int width)
  {
  } /* end mdlSetInputPortWidth */
#endif /* MDL_SET_INPUT_PORT_WIDTH */


#define MDL_SET_OUTPUT_PORT_WIDTH   /* Change to #undef to remove function */
#if defined(MDL_SET_OUTPUT_PORT_WIDTH) && defined(MATLAB_MEX_FILE)
  /* Function: mdlSetOutputPortWidth
   * Abstract:
   *   输出端口个数动态变化时调用
   */
  static void mdlSetOutputPortWidth(SimStruct *S, int portIndex, int width)
  {
```

```
} /* end mdlSetOutputPortWidth */
#endif /* MDL_SET_OUTPUT_PORT_WIDTH */


#undef MDL_SET_INPUT_PORT_DIMENSION_INFO /* Change to #define to add function
*/
#if         defined(MDL_SET_INPUT_PORT_DIMENSION_INFO)        &&
defined(MATLAB_MEX_FILE)
  /* Function: mdlSetInputPortDimensionInfo
   * Abstract:
   *    输入端口维数未知时调用
   */
  static void mdlSetInputPortDimensionInfo(SimStruct        *S,
                                           int_T            portIndex,
                                           const DimsInfo_T *dimsInfo)
  {
  } /* mdlSetInputPortDimensionInfo */
#endif /* MDL_SET_INPUT_PORT_DIMENSION_INFO */


#undef MDL_SET_OUTPUT_PORT_DIMENSION_INFO /* Change to #define to add
function*/
#if defined(MDL_SET_OUTPUT_PORT_DIMENSION_INFO) && defined(MATLAB_
MEX_FILE)
  /* Function: mdlSetOutputPortDimensionInfo
   * Abstract:
   *  输出端口维数未知时调用
   */
  static void mdlSetOutputPortDimensionInfo(SimStruct        *S,
                                            int_T            portIndex,
                                            const DimsInfo_T *dimsInfo)
  {
  } /* mdlSetOutputPortDimensionInfo */
#endif /* MDL_SET_OUTPUT_PORT_DIMENSION_INFO */

#undef MDL_SET_DEFAULT_PORT_DIMENSION_INFO /* Change to #define to add fcn
*/
#if         defined(MDL_SET_DEFAULT_PORT_DIMENSION_INFO)        &&
defined(MATLAB_MEX_FILE)
  /* Function: mdlSetDefaultPortDimensionInfo
```

```
 * Abstract:
 *     无法确定端口维数时调用
 */
 static void mdlSetDefaultPortDimensionInfo(SimStruct *S)
 {
 } /* mdlSetDefaultPortDimensionInfo */
#endif /* MDL_SET_DEFAULT_PORT_DIMENSION_INFO */


#define MDL_SET_INPUT_PORT_SAMPLE_TIME
#if  defined(MDL_SET_INPUT_PORT_SAMPLE_TIME)  &&  defined(MATLAB_MEX_FILE)
 /* Function: mdlSetInputPortSampleTime
 * Abstract:
 *     设置输入端口采样时间
 */
 static void mdlSetInputPortSampleTime(SimStruct *S,
                                       int_T     portIdx,
                                       real_T    sampleTime,
                                       real_T    offsetTime)
 {
 } /* end mdlSetInputPortSampleTime */
#endif /* MDL_SET_INPUT_PORT_SAMPLE_TIME */

#define MDL_SET_OUTPUT_PORT_SAMPLE_TIME
#if  defined(MDL_SET_OUTPUT_PORT_SAMPLE_TIME)  &&  defined(MATLAB_MEX_FILE)
 /* Function: mdlSetOutputPortSampleTime
 * Abstract:
 *     设置输入端口采样时间
 */
 static void mdlSetOutputPortSampleTime(SimStruct *S,
                                        int_T     portIdx,
                                        real_T    sampleTime,
                                        real_T    offsetTime)
 {
 } /* end mdlSetOutputPortSampleTime */
#endif /* MDL_SET_OUTPUT_PORT_SAMPLE_TIME */
```

```
/* Function: mdlInitializeSampleTimes
 * Abstract:
 *
 *     确定系统的采样时间
 *
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    /* Register one pair for each sample time */
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);

} /* end mdlInitializeSampleTimes */

#define MDL_SET_INPUT_PORT_DATA_TYPE    /* Change to #undef to remove function
*/
#if defined(MDL_SET_INPUT_PORT_DATA_TYPE) && defined(MATLAB_MEX_FILE)
  /* Function: mdlSetInputPortDataType
   * Abstract:
   *     输入端口数据类型动态变化时调用
   */
  static void mdlSetInputPortDataType(SimStruct *S, int portIndex,DTypeId dType)
  {
  } /* mdlSetInputPortDataType */
#endif /* MDL_SET_INPUT_PORT_DATA_TYPE */

#define  MDL_SET_OUTPUT_PORT_DATA_TYPE   /*  Change  to  #undef  to  remove
function */
#if  defined(MDL_SET_OUTPUT_PORT_DATA_TYPE)  &&  defined(MATLAB_MEX_
FILE)
  /* Function: mdlSetOutputPortDataType
   * Abstract:
   *      输出端口数据类型动态变化时调用
   */
  static void mdlSetOutputPortDataType(SimStruct *S,int portIndex,DTypeId dType)
  {
  } /* mdlSetOutputPortDataType */
#endif /* MDL_SET_OUTPUT_PORT_DATA_TYPE */

#define  MDL_SET_DEFAULT_PORT_DATA_TYPES  /*  Change  to  #undef  to  remove
```

```
function*/
    #if defined(MDL_SET_DEFAULT_PORT_DATA_TYPES) && defined(MATLAB_MEX_
FILE)
      /* Function:   mdlSetDefaultPortDataTypes
       * Abstract:
       *     无法确定端口数据类型时调用
       */
      static void mdlSetDefaultPortDataTypes(SimStruct *S)
      {
      } /* mdlSetDefaultPortDataTypes */
    #endif /* MDL_SET_DEFAULT_PORT_DATA_TYPES */

    #define MDL_SET_INPUT_PORT_COMPLEX_SIGNAL    /* Change to #undef to remove
*/
    #if  defined(MDL_SET_INPUT_PORT_COMPLEX_SIGNAL)  &&  defined(MATLAB_
MEX_FILE)
      /* Function: mdlSetInputPortComplexSignal
       * Abstract:
       *     设置复信号端口
       */
      static void mdlSetInputPortComplexSignal(SimStruct *S,
                                               int          portIndex,
                                               CSignal_T cSignalSetting)
      {
      } /* mdlSetInputPortComplexSignal */
    #endif /* MDL_SET_INPUT_PORT_COMPLEX_SIGNAL */

    #define  MDL_SET_OUTPUT_PORT_COMPLEX_SIGNAL    /*  Change  to  #undef  to
remove */
    #if  defined(MDL_SET_OUTPUT_PORT_COMPLEX_SIGNAL)  &&  defined(MATLAB_
MEX_FILE)
      /* Function: mdlSetOutputPortComplexSignal
       * Abstract:
       *     对属性为 COMPLEX_INHERITED 的复信号进行设置
       */
      static void mdlSetOutputPortComplexSignal(SimStruct *S,
                                                int          portIndex,
                                                CSignal_T cSignalSetting)
      {
      } /* mdlSetOutputPortComplexSignal */
```

```
#endif /* MDL_SET_OUTPUT_PORT_COMPLEX_SIGNAL */

#define MDL_SET_DEFAULT_PORT_COMPLEX_SIGNALS /* Change to #undef to
remove */
#if defined(MDL_SET_DEFAULT_PORT COMPLEX_SIGNALS) && defined(MATLAB_
MEX_FILE)
  /* Function:  mdlSetDefaultPortComplexSignals
   * Abstract:
   *     无法确定复信号时调用
   */
  static void mdlSetDefaultPortComplexSignals(SimStruct *S)
  {
  } /* mdlSetDefaultPortComplexSignals */
#endif /* MDL_SET_DEFAULT_PORT_COMPLEX_SIGNALS */

#define MDL_SET_WORK_WIDTHS    /* Change to #undef to remove function */
#if defined(MDL_SET_WORK_WIDTHS) && defined(MATLAB_MEX_FILE)
  /* Function: mdlSetWorkWidths
   * Abstract:
   *       工作向量为 DYNAMICALLY_SIZED 时设置其维数
   */
  static void mdlSetWorkWidths(SimStruct *S)
  {
  }
#endif /* MDL_SET_WORK_WIDTHS */

#define MDL_INITIALIZE_CONDITIONS    /* Change to #undef to remove function */
#if defined(MDL_INITIALIZE_CONDITIONS)
  /* Function: mdlInitializeConditions
   * Abstract:
   * 初始化连续离散模块
   */
  static void mdlInitializeConditions(SimStruct *S)
  {
  }
#endif /* MDL_INITIALIZE_CONDITIONS */


#define MDL_START   /* Change to #undef to remove function */
#if defined(MDL_START)
```

```
/* Function: mdlStart
 * Abstract:
 *     对只需要初始化一次的模块进行初始化
 */
static void mdlStart(SimStruct *S)
{
}
#endif /*   MDL_START */


#define MDL_GET_TIME_OF_NEXT_VAR_HIT   /* Change to #undef to remove function
*/
#if defined(MDL_GET_TIME_OF_NEXT_VAR_HIT) && (defined(MATLAB_MEX_FILE)
||\
                                              defined(NRT))
/* Function: mdlGetTimeOfNextVarHit
 * Abstract:
 *     得到采样时间的下一个时间步
 */

static void mdlGetTimeOfNextVarHit(SimStruct *S)
{
    time_T timeOfNextHit=ssGetT(S) /*+offset */ ;
    ssSetTNext(S, timeOfNextHit);
}
#endif /* MDL_GET_TIME_OF_NEXT_VAR_HIT */


#define MDL_ZERO_CROSSINGS     /* Change to #undef to remove function */
#if  defined(MDL_ZERO_CROSSINGS)  &&  (defined(MATLAB_MEX_FILE)  ||  defined
(NRT))
/* Function: mdlZeroCrossings
 * Abstract:
 *    处理过零点
 */
static void mdlZeroCrossings(SimStruct *S)
{
}
#endif /* MDL_ZERO_CROSSINGS */
```

```
/* Function: mdlOutputs
 * Abstract:
 *     计算输出
 */
static void mdlOutputs(SimStruct *S, int_T tid)
{
} /* end mdlOutputs */


#define MDL_UPDATE   /* Change to #undef to remove function */
#if defined(MDL_UPDATE)
  /* Function: mdlUpdate
   * Abstract:
   *     离散系统中迭代用
   */
  static void mdlUpdate(SimStruct *S, int_T tid)
  {
  }
#endif /* MDL_UPDATE */


#define MDL_DERIVATIVES   /* Change to #undef to remove function */
#if defined(MDL_DERIVATIVES)
  /* Function: mdlDerivatives
   * Abstract:
   *  求微分
   */
  static void mdlDerivatives(SimStruct *S)
  {
  }
#endif /* MDL_DERIVATIVES */


/* Function: mdlTerminate
 * Abstract:
 *  退出时进行清理工作
 */
static void mdlTerminate(SimStruct *S)
{
```

```
}
#define MDL_RTW   /* Change to #undef to remove function */
#if defined(MDL_RTW) && defined(MATLAB_MEX_FILE)
  /* Function: mdlRTW
   * Abstract:
   *产生*.rtw 文件时调用
   */
   static void mdlRTW(SimStruct *S)
  {
  }
#endif /* MDL_RTW */
/*=================================*
 * Required S-function trailer *
 *=================================*/

#ifdef   MATLAB_MEX_FILE      /* Is this file being compiled as a MEX-file? */
#include "simulink.c"         /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"          /* Code generation registration function */
#endif
```

mdlInitializeSizes 是 Simulink 与 S 函数交互时调用的第一个子程序。S 函数中还有其他一些*.mdl 子程序，所有 S 函数必须依照这样的格式，这就意味着不同的 S 函数可以使用相同名字、不同内容的 S 函数子程序来实现。在调用 mdlInitializeSize 子程序后，Simulink 再与其他各个子程序交互，仿真最后，调用 mdlTerminate 子程序。

Mex 文件必须定义具体的函数，这些函数既提供了执行这些任务需要的信息，也包含了另外必须的一些代码，表 8-6 描述了 Simulink 在仿真期间调用的一些函数，它们是按调用的先后顺序给出的，任一 S 函数 Mex 文件必须包含所有这些函数，即使一个模型并不一定用到所有这些函数。

表 8-6　　Simulink 在仿真期间调用的部分函数

仿真阶段	S 函数子程序
对模块（输入输出向量的）大小信息、采样时间和初始状态的初始化	mdlInitializeSizes mdlInitializeSampleTimes mdlInitializeConditions
计算各个输出	mdlOutputs
更新各个离散状态	mdlUpdate
计算下一个采样时间点（可选）	mdlGetTimeOfNextVarHit
计算各个连续状态的导数	mdlDerivatives
在仿真结束时执行的各项任务	mdlTerminate

与 M 文件形式的 S 函数不同的是，C Mex 文件形式的 S 函数没有与每一个 S 函数子程序相联系的 flag 参数，这是因为 Simulink 在各个仿真阶段的合适的时候会自动地调用各个 S

函数子程序。同样，C Mex 文件形式的 S 函数还有一些 S 函数子程序，而 M 文件形式的 S 函数没有与它们相对应的子程序，这些子程序包括 mdlInitializeSampleTimes 和 mdlInitializeConditions。

Simulink 用一个被称为 SimStruct 的数据结构维护 S 函数模块的有关信息。定义 SimStruct 的 include 文件 simstruc.h 提供了使得 Mex 文件可以从 SimStruct 中设置和取得值的宏。

Simulink 提供了一个 C Mex 文件形式的 S 函数模板，里面包含有定义一些必要的功能的语句和一些注释，这些语句有助于编写 S 函数模块中所必需的代码，这一模板文件 sfuntmpl_basic.c 或 sfuntmpl_doc.c（这个更详细）可以在 MATLAB 根目录下的 Simulink/src 目录中找到。建议在开发 C Mex 文件形式的 S 函数时使用 C Mex 文件正式的模板。

8.3.2 C Mex 文件形式的 S 函数例子

MATLAB 提供了与 8.2 节中 M 文件形式的 S 函数相对应的例子，在此限于篇幅不再赘述，读者可以自己到 MATLAB 根目录下的 Simulink/src 目录中找到。

8.3.3 使用 Function-Call 子系统

可以创建一个可触发子系统，它的执行取决于一个 S 函数的内部逻辑而不是信号的值。一个如此设定的子系统被称为 function-call 子系统。要实现一个 function-call 子系统，按如下的步骤进行：

（1）在 Trigger 模块中，选择 function-call 作为 Trigger type 参数。

（2）在 S 函数中，使用 ssCallSystemWithTid 宏去调用可触发子系统。

（3）在模型中，将 S 函数模块的输出直接连到触发端口。

Function-call 子系统并不是直接被 Simulink 执行，S 函数确定何时执行该子系统，当子系统执行完毕，控制返回给 S 函数。

Function-call 子系统只能够连接到已经正确地配置了、可以接受它们的 S 函数。要设定一个 S 函数，可以调用一个 Function-call 子系统，按如下步骤进行：

（1）在 mdlInitializeSampleTimes 中指定哪些元素将用来执行 function-call 系统。如：

```
ssSetCallSystemOutput(S,0);/* call on 1st element */
ssSetCallSystemOutput(S,2);/* call on 3rd element */
```

（2）在适当的 mdlOutputs 或 mdlUpdates 子程序中执行该子系统。例如：

```
static void mdlOutputs(SimStruct *S, int_T tid)
{
    real_T              *x    =ssGetRealDiscStates(S);
    InputRealPtrsType uPtrs=ssGetInputPortRealSignalPtrs(S,0);
    real_T              *y    =ssGetOutputPortRealSignal(S,1);

    /*
     * ssCallSystemWithTid is used to execute a function-call subsystem. The
     * 2nd argument is the element of the 1st output port index which
```

```
 * connected to the function-call subsystem. Function-call subsystems
 * can be driven by the first output port of s-function blocks.
 */

 UNUSED_ARG(tid); /* not used in single tasking mode */

 if (((int)*uPtrs[0]) % 2 == 1) {
   if (!ssCallSystemWithTid(S,0,tid)) {
     /* Error occurred which will be reported by Simulink */
     return;
   }
 } else {
    if (!ssCallSystemWithTid(S,1,tid)) {
      /* Error occurred which will be reported by Simulink */
      return;
   }
 }
 y[0]=x[0];
}
```

在 sfun_fcncall.c 中演示了一个 S 函数被设定成可执行 function-call 子系统。Function-call 子系统通常被用于 Stateflow 模块。

8.3.4 S 函数类型

（1）非嵌入 S 函数

非嵌入 S 函数是一个 C Mex 函数，在 Simulink 和 RTW 中同样对待。一般情况下，以前是根据 S 函数 API 来实现算法的。而 Simulink 和 RTW 调用 S 函数子程序是在模型执行的合适的点上。每个非嵌入 S 函数模块实例都需要重要的内存和计算资源，这种综合 Simulink 和 RTW 的算法子程序一般在原型设计阶段使用，此时效率并不重要。其优点是带来了快速改变模型参数和结构的能力。

写非嵌入 S 函数不包含 TLC 代码，非嵌入 S 函数是 RTW 的缺省情况，一旦在模型中创建了 S 函数，在模型的仿真参数对话框的 RTW 页面中按 Build 之前，无需其他准备工作。

对于非嵌入 S 函数的执行，需要 C Mex S 函数的源代码（sfunction.c）。

（2）包装 S 函数

包装 S 函数对于连接手写代码或包装在一些程序中的大算法是非常理想的。这种情况下，通常这些程序驻留在与 C Mex S 函数分离的模块中。S 函数模块通常包含对这些程序的一些调用。由于 S 函数不包含这些算法的任何部分，仅调用其代码，因此叫做包装 S 函数。除了包装 C Mex 函数的包装外，还需要创建 TLC 包装来实现 S 函数，TLC 包装与 S 函数包装一样，其中包含有算法程序。

对于嵌入调用算法（C 语言函数）的包装 S 函数的执行，需要一个 sfunction.tlc 文件。

（3）完全嵌入 S 函数

完全嵌入 S 函数将以一种与内置模块没有区别的方式，在 Simulink 和 RTW 中创建算法（模块）。一般地，完全嵌入 S 函数需要执行算法两次：一是为了 Simulink（C Mex S 函数），二是为了 TRW（TLC 文件）。TLC 文件的复杂性取决于算法的复杂性，以及产生的代码要达到的效率水平。TLC 文件结构有简单，有复杂。对于完全嵌入式 S 函数的执行，需要一个 sfunction.tlc 文件。

对于使用 RTW 的 S 函数子程序的完全嵌入式 S 函数，需要放置 mdlRTW 子程序于 S 函数 Mex 文件 sfunction.c 中，mdlRTW 子程序使得用户可以放置信息于 model.rtw 中。这是一个在产生代码和导入非可调参数时，在执行 sfunction.tcl 之前，用目标语言编译器处理的文件。

第9章　Simulink6.0在信号处理仿真中的应用

本章主要介绍信号处理仿真的基础知识，以及运用 Simulink 进行仿真的方法，介绍 Simulink 中常用的信号仿真模块，然后通过实例说明如何把这些知识应用于实践。

本章主要内容：

- ➢ 信号处理仿真基础
- ➢ Simulink6.0 中的数字信号处理仿真模块
- ➢ 通信系统仿真实例 1——信号滤波
- ➢ 通信系统仿真实例 2——卡尔曼滤波

9.1　信号处理仿真基础

Simulink 提供了强大的信号处理工具，这些工具使信号仿真工作，与那种自己写代码实现算法的仿真相比，变得非常轻松。在此不对信号处理的基本知识进行赘述，这些知识读者可以参考其他信号与系统的参考书。此处仅对 Simulink 仿真的一些基础知识做介绍。

Simulink 信号处理工具包对信号的处理基于两种基本单元。一种是基于采样信号，另一种是基于帧的信号。基于采样的信号大家都比较熟悉，此处介绍一下基于帧的信号。

大多数实时的数字信号处理系统都采用基于帧的处理方式，以提高系统性能，这里每帧包含相邻的多个或者一组信号采样。采用基于帧的处理方式更适合多数的数字信号处理算法，另外也可降低系统对数据采集硬件的要求。缺省情况下，Simulink 所有信号都是基于采样的。

基于采样的信号和基于帧的信号的区别如表 9-1 所示。

表 9-1

基于采样的信号	基于帧的信号
每个时间步处理一个采样点	每个时间步处理含有 N 个采样点的一帧
仿真步长=采样周期=Ts	仿真步长=帧周期=$N*Ts$
采样频率 $Fs=1/Ts$	帧频率=Fs/N
采样步长可变	帧的大小可变，可以是工作区中的一个变量

之所以采用基于帧的处理主要是考虑到数字信号处理本身的要求和数据通信的开销。显然，基于帧的信号处理应当比基于采样的处理要复杂得多，但是 Simulink 利用 MATLAB 的矩阵功能极大地提高了处理的效率。通过基于帧的处理，减少了块与块之间的通信，从而比使用基于采样的信号进行仿真快得多。总之，利用基于帧的信号提高了仿真速度。而且，由于同样的原因，大多数 DSP 系统也采用基于帧的处理。除此之外，基于帧的处理提供了在仿

真中进行频域分析的能力。

Simulink 的所有模块都支持基于帧的处理，使得用户可以方便地采用基于帧的信号进行算法仿真以及结合 RTW 产生实时代码。

图 9-1 说明了从连续信号经过 AD 采样得到采样信号，然后将采样信号组织成帧，送往 Simulink 处理的过程。

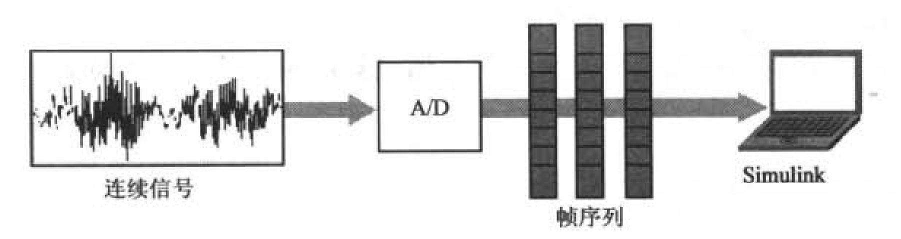


图 9-1 信号处理过程

1. 缓冲和解缓冲

在 Simulink 中采样信号和帧信号之间的转换是通过缓冲模块（Buffer）来实现的。Buffer 块有两种用途：一是接受采样输入并产生一定帧大小的帧输入；二是接受帧输入，修改帧的大小，这种情况下必须使用缓冲模块。这两种情况下都涉及帧之间的重叠和帧的初始值的设置问题。当通过采样产生帧时，缓冲使用输入标量生成一个列向量，如图 9-2 所示。如果需要从一个帧信号产生一个采样信号，则应使用 Unbuffer 模块。

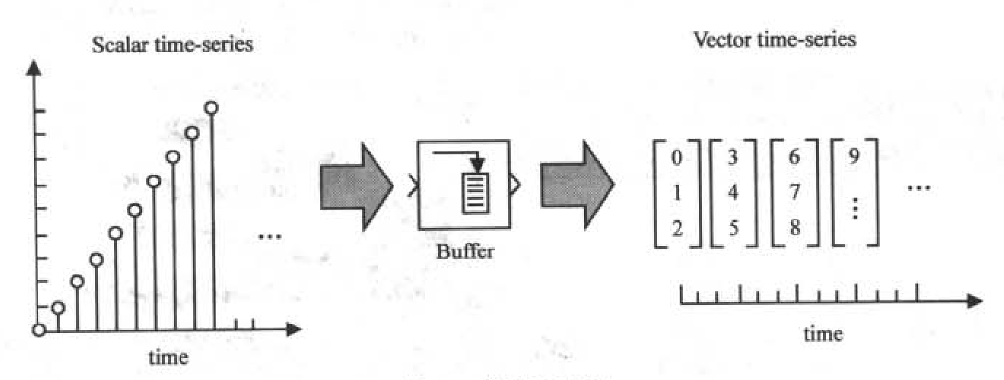


图 9-2 缓冲和解缓冲

Source 库中的许多信号源模块同样提供基于帧的输出，当然使用这些模块作为输入信号时，就无需使用 Buffer 块，只需设置块的帧长参数就可以了。

2. 帧的表示

通常，一帧是通过一个矩阵表示的。在帧矩阵中，每个通道的信号对应矩阵中的一列，每个采样对应其中的一行（如图 9-3 所示）。在基于帧的处理中，各个模块沿着输入的每一列（通道）进行运算。图 9-3 中有四个信号通道，每帧有两个采样，帧和帧之间没有重叠。通常每帧的采样数是 2 的幂次，以满足 FFT 变换的需要。

Ch：通道
Sample：采样
Frame：帧

	Ch1	Ch2	Ch3	Ch4	
Sample1	1	2	3	4	Frame1
Sample2	2	3	4	5	
Sample3	3	4	5	6	Frame2
Sample4	4	5	6	7	
Sample5	5	6	7	8	Frame3
Sample6	6	7	8	9	

图 9-3 帧的表示

3. 生成基于帧的信号

主要有三种方法用来生成基于帧的多通道信号。

（1）Signal Processing 模块库中信号源库 Signal Processing Sources 中的块提供了信号源块，用于生成基于帧的信号。

（2）所有的信号都可以通过缓冲块成为帧。

（3）将从若干个基于帧的信号源来的信号通过矩阵拼接成一个帧矩阵，形成一个多路信号。

4. 观察基于帧的信号

用户可以使用 Signal Processing blockset 提供的专门的显示模块来观察基于帧的信号。这些模块中最常用的是 Matrix Viewer（矩阵浏览器）和 Vector Scope（向量示波器）。Matrix Viewer 将输入矩阵的行和列作为坐标轴，使用不同颜色表示矩阵元素的值，还可以根据需要自己建立一个颜色表。Vector Scope 显示输入的每一列（通道），按照指定帧的数目每次显示整个数据。Vector Scope 可以显示时域或频域信号。图 9-4 是基于帧的三个正弦信号（三个通道）分别用 Matrix Viewer 和 Vector Scope 显示的结果。此外还有内置 FFT 变换的 Spectrum Scope 用来直接显示时域信号的频谱。

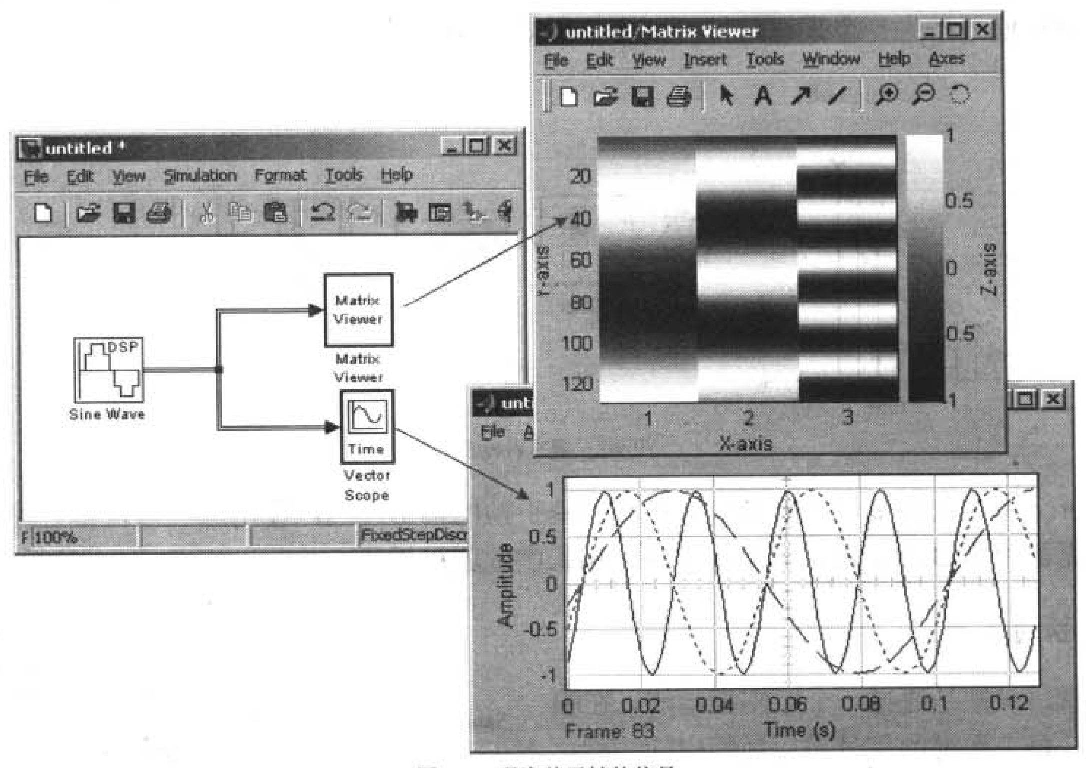


图 9-4 观察基于帧的信号

5. 使用基于帧的信号

当一个信号线表示基于帧的信号时，Simulink 用双线来绘制。基于帧的信号处理可以使用 Simulink 中对输入的每个元素进行处理的块，但是不能使用 Simulink 中对向量处理的模块（例如 Unit Delay 和 Mux）。实际上这些模块中许多模块在 Signal Processing Blockset 中都有一

个与之对应，专门用来做基于帧的信号处理的版本。例如，在 Signal Processing Blockset 中等价于 Unit Delay 的模块是 Integer Delay 模块，与 Mux 等价的模块是 Matrix Concatenation 模块。图 9-5 所示的框图是对随机信号延迟 30 个步长后进行卷积处理。

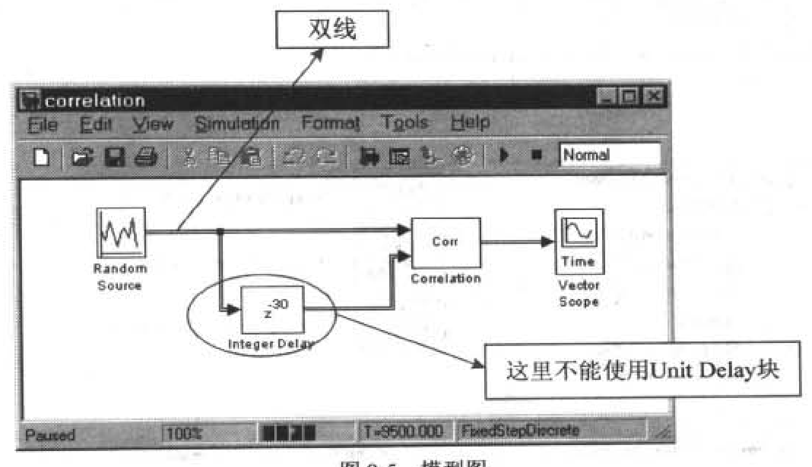


图 9-5 模型图

9.2 Simulink6.0 中数字信号处理仿真模块

Simulink6.0 提供了丰富的数字信号处理模块，几乎包括了信号处理所用到的所有操作和算法。如图 9-6 所示。其中的许多模块在信号处理工具箱中都有对应的函数。用户可以利用这些模块方便地完成自己的数字信号处理系统仿真和分析，这一节将分别介绍各个子库。

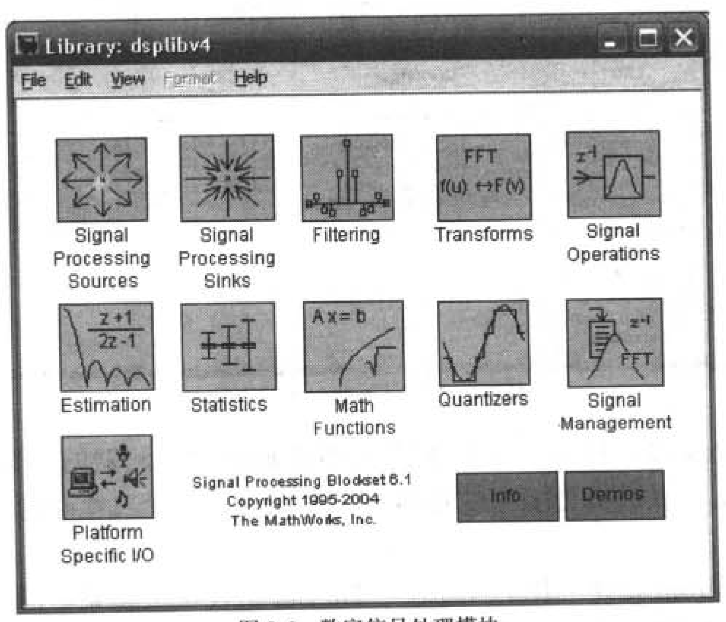


图 9-6 数字信号处理模块

9.2.1 Estimation 子模块集

Estimation 子模块集有如下三个大类 Linear Prediction 模块组、Parametric Estimation 模块

组、Power Spectrum Estimation 模块。如图 9-7 所示。

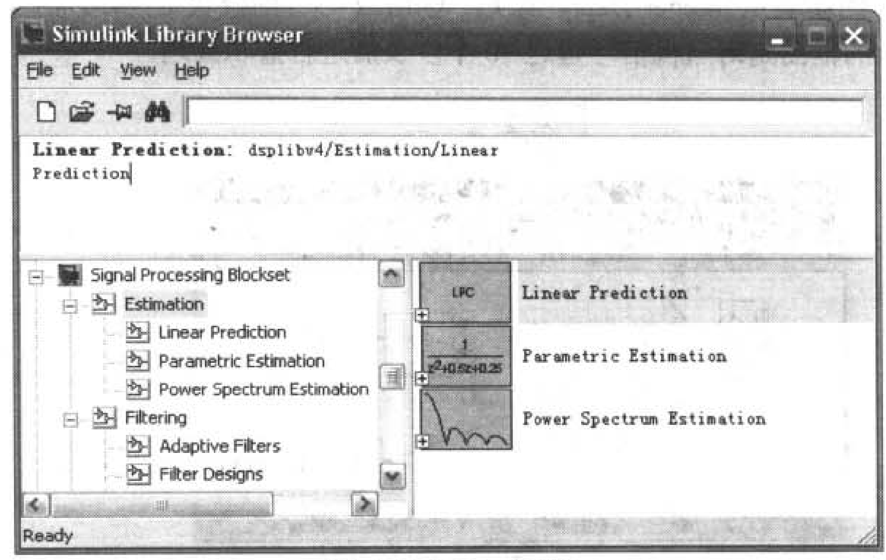


图 9-7 Estimation 子模块

- Linear Prediction 模块组包含以下模块，如图 9-8 所示，分别是不同的线形预测模块。

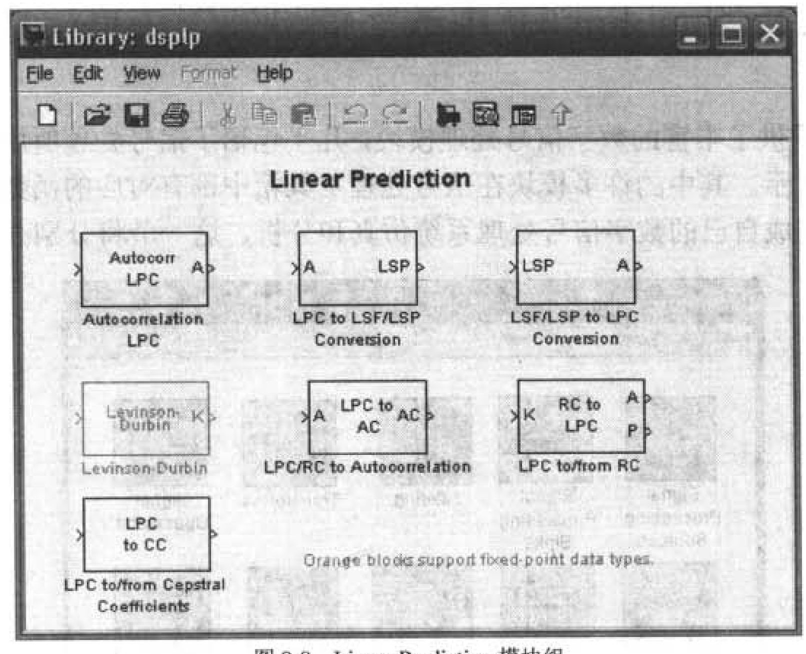


图 9-8 Linear Prediction 模块组

> Autocorrelation LPC 模块　用来求解 N 阶前向线形预测器的系数，输入为长度为 M 的向量，输出可以选择为多项式系数或反射系数，或者二者都输出，还可以输出误差能量。

> Levinson－Durbin 模块　用 Levinson－Durbin 迭代法求解系数满足 Toeplitz 矩阵的系统。输出可以选择为多项式系数或反射系数，或者二者都输出，还可以输出误差能量。

> LPC to LSF/LSP Conversion 把 LPC 系数转换为 LSF 或 LSP，注意输入的第一个系数最好为 1，若不是 1，则模块会用默认的方法单位化，同时给出警告。

- Parametric Estimation 模块组包含以下模块，如图 9-9 所示，分别是不同的参数估计模块。

- Power Spectrum Estimation 模块组包含以下模块，如图 9-10 所示，分别是不同的功率谱估计模块。

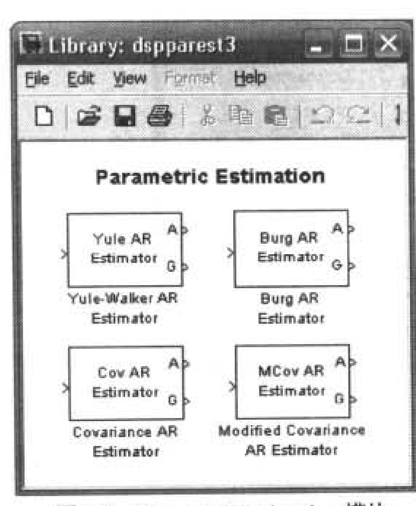


图 9-9 Parametric Estimation 模块

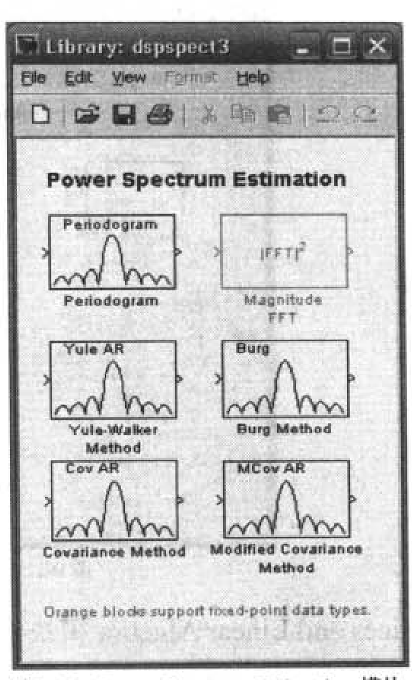


图 9-10 Power Spectrum Estimation 模块

9.2.2 Math Function 子模块集

Math Function 子模块集有如下三个大类：Math Operation 模块组、Matrices and Linear Algebra 模块组、Polynomial Functions 模块组，如图 9-11 所示。

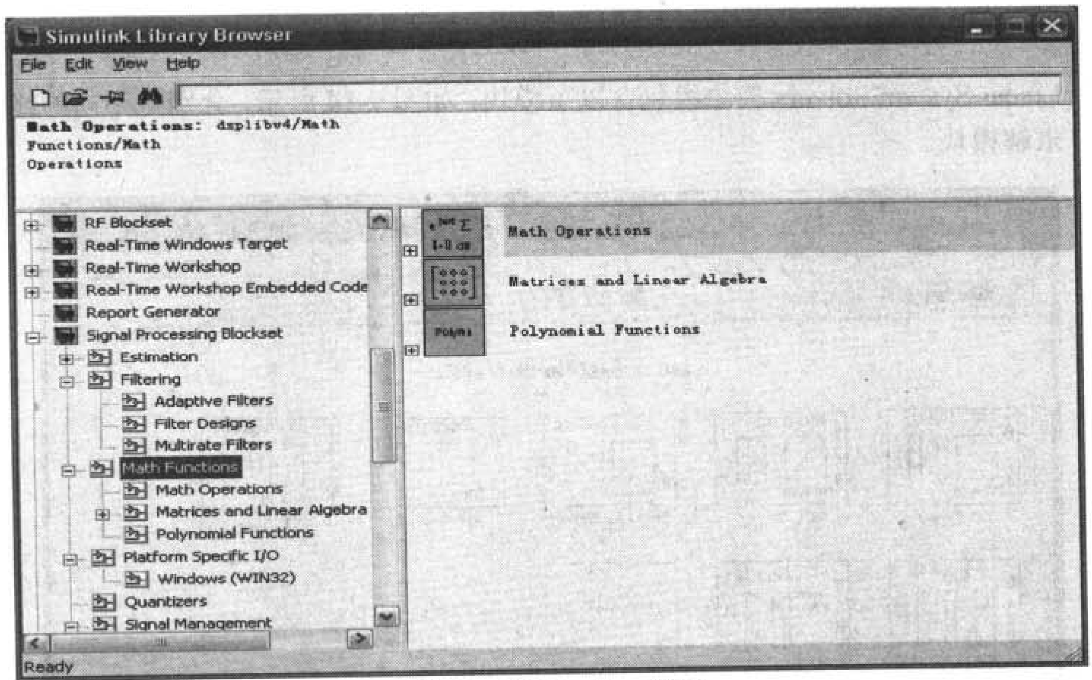


图 9-11 Math Function 子模块

- Math Operation 模块组包含以下模块，如图 9-12 所示，分别是不同的数学运算模块，

包括复指数、累乘、累加、分贝转换，单位化等操作。

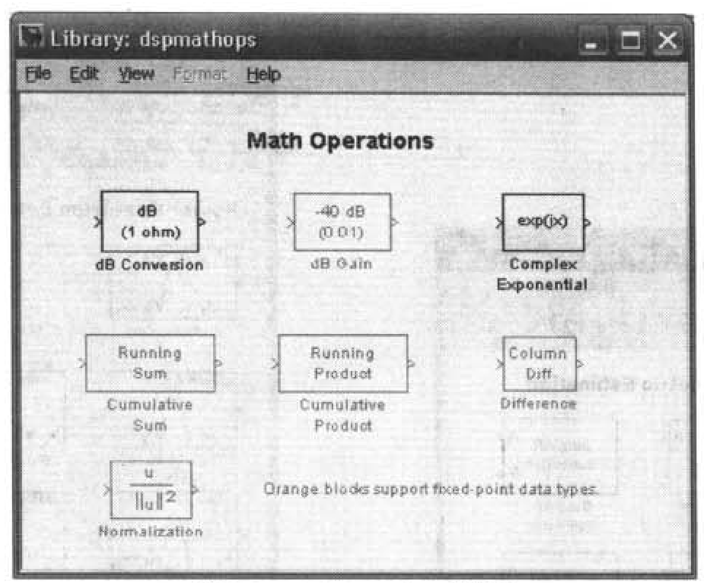


图 9-12　Math Operation 模块组

- Matrices and Linear Algebra 模块组包含以下四个子模块组，如图 9-13 所示。

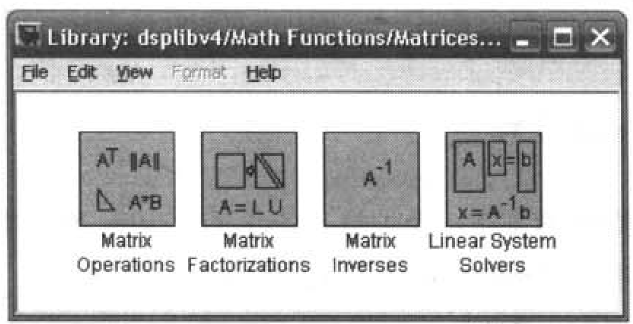


图 9-13　Matrices and Linear Algebra 模块组

> Linear System Solvers 模块组包含以下模块，如图 9-14 所示，分别是不同的线性方程求解模块。

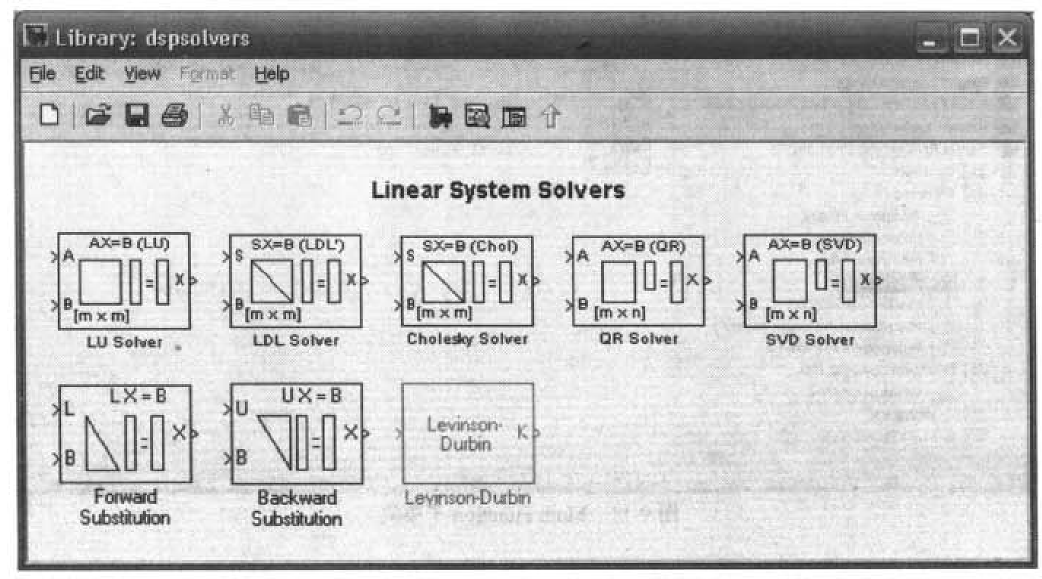


图 9-14　Linear System Solvers 模块组

➢ Matrix Factorizations 模块组包含以下模块，如图 9-15 所示，分别是不同的矩阵分解模块。

➢ Matrix Inverses 模块组包含以下模块，如图 9-16 所示，分别是不同的矩阵求逆模块。

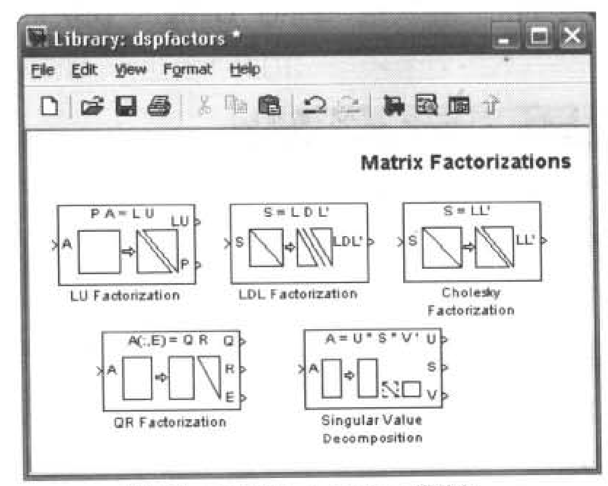


图 9-15 Matrix Factorizations 模块组

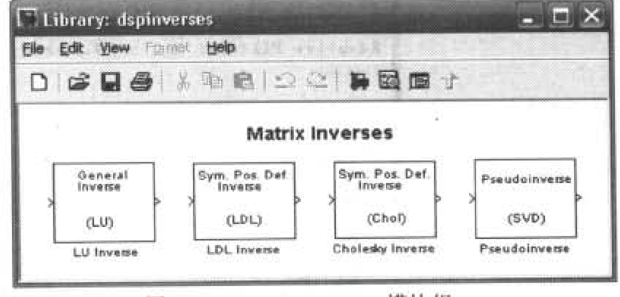


图 9-16 Matrix Inverses 模块组

➢ Matrix Operations 模块组包含以下模块，如图 9-17 所示，分别是不同的矩阵操作模块。

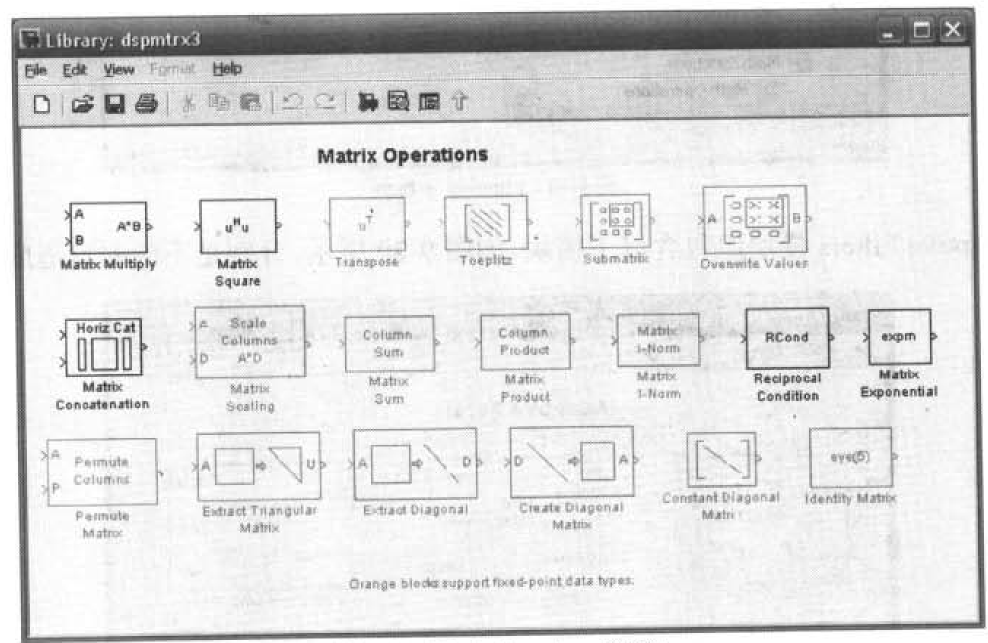


图 9-17 Matrix Operations 模块组

• Polynomial Functions 模块组包含以下模块，如图 9-18 所示，分别是不同的多项式操作模块。

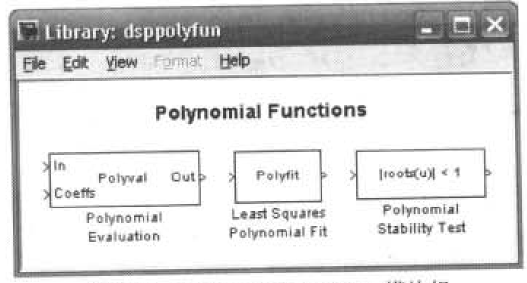


图 9-18 Polynomial Functions 模块组

9.2.3 Filtering 子模块集

Filtering 子模块集有如下三个大类：Adaptive Filter 模块组、Filter Designs 模块组、Multirate Filters 模块组，如图 9-19 所示。

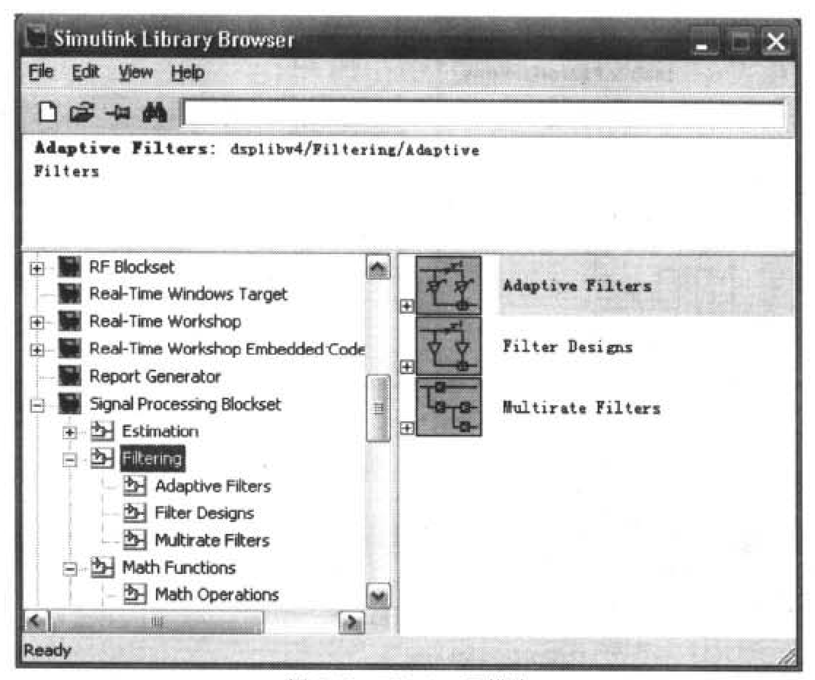


图 9-19 Filtering 子模块

- Adaptive Filters 模块组包含以下模块，如图 9-20 所示，分别是不同的自适应滤波模块。

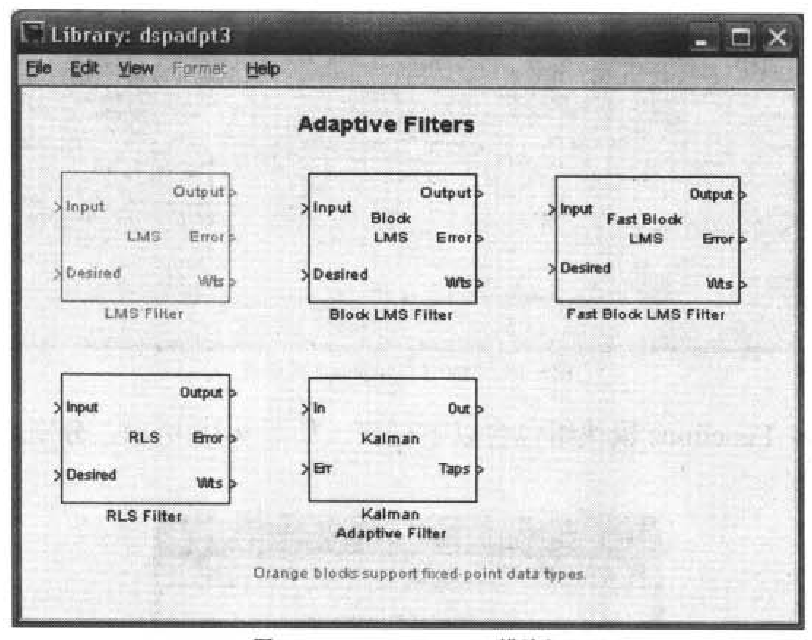


图 9-20 Adaptive Filters 模块组

- Filter Designs 模块组包含以下模块，如图 9-21 所示，分别是不同的滤波器设计模块。
- Multirate Filters 模块组包含以下模块，如图 9-22 所示，分别是不同的多速率滤波模块。

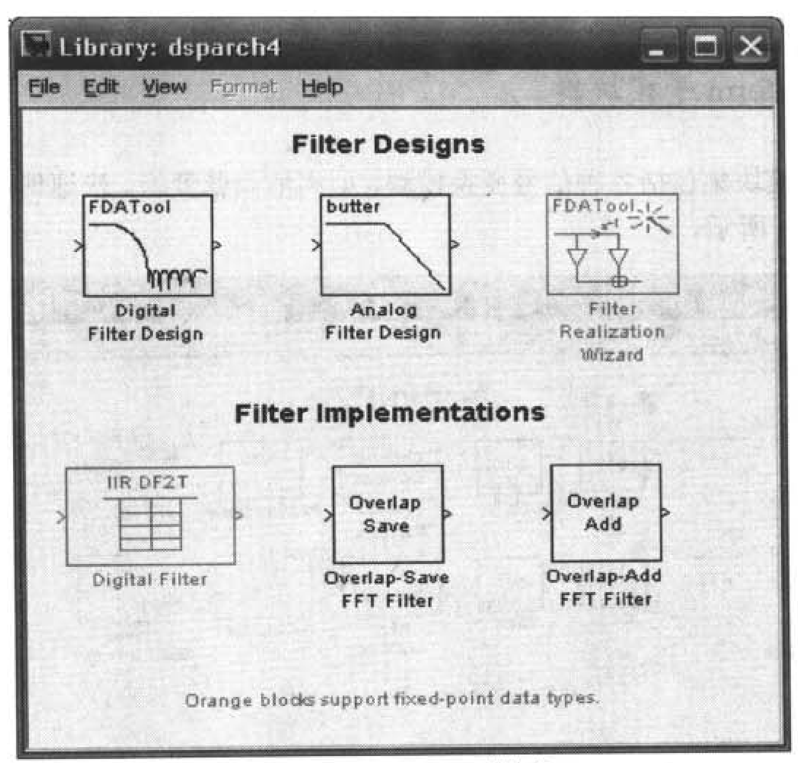


图 9-21 Filter Designs 模块组

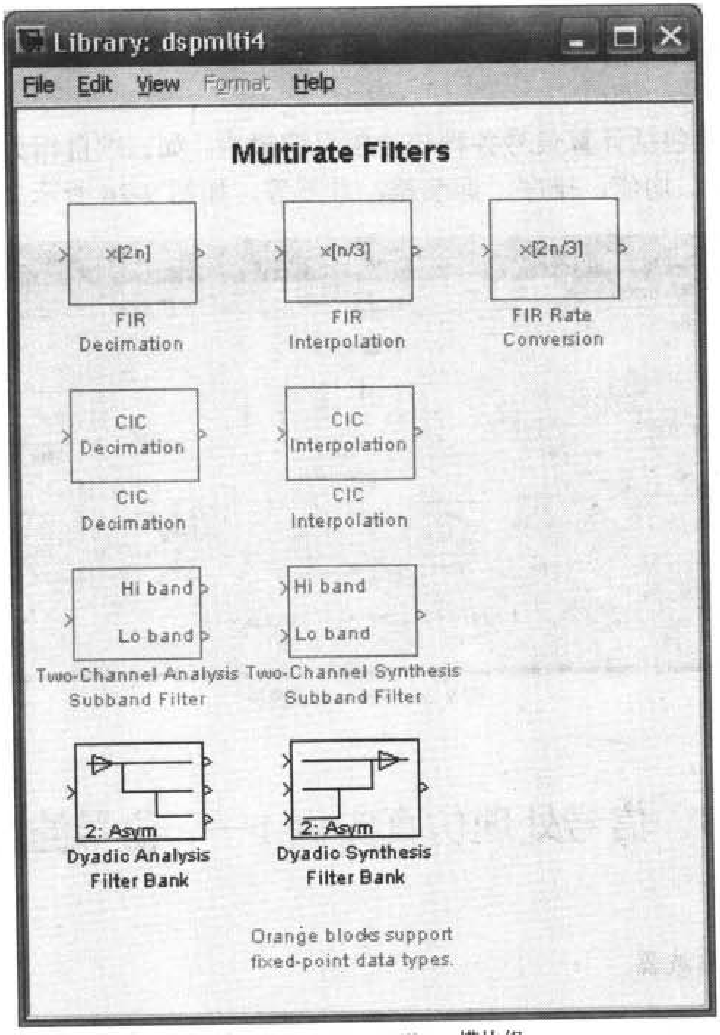


图 9-22 Multirate Filters 模块组

9.2.4 Transform 子模块集

Transform 子模块集包括各种信号变换模块，如离散余弦变换、快速傅立叶变换及相应的逆变换，如图 9-23 所示。

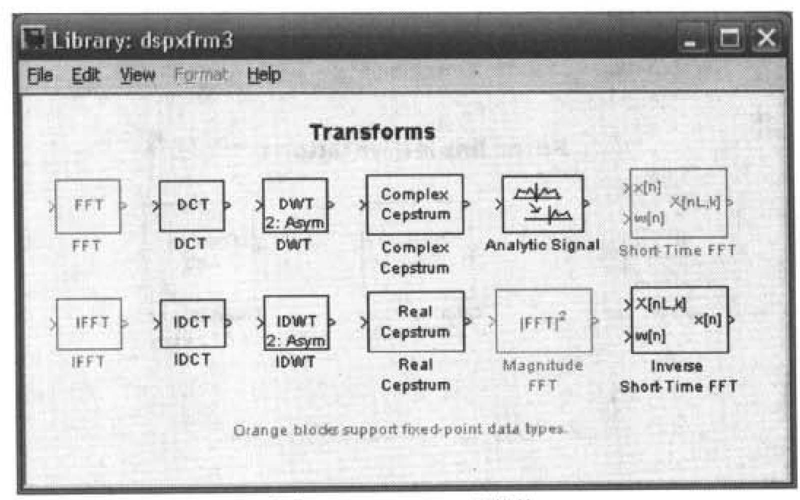


图 9-23 Transform 子模块

9.2.5 Statistic 子模块集

Statistic 子模块集包括计算信号各种统计信息的模块，如：求自相关、协方差、统计直方图、最大值、最小值、均值、排序、标准差、方差等，如图 9-24 所示。

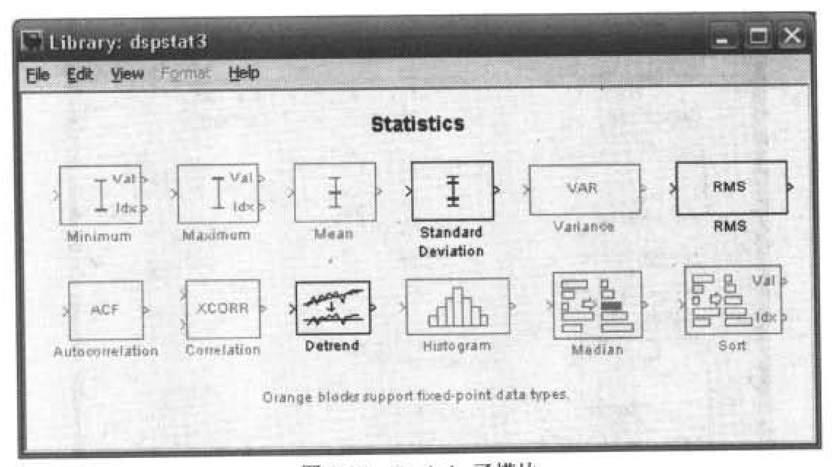


图 9-24 Statistic 子模块

9.3 信号处理仿真实例 1——信号滤波

1. LMS 自适应滤波器

最小均方误差（LMS）自适应滤波器是采用最小均方误差准则的自适应滤波器，所谓自

适应，通俗地讲，就是滤波器能够根据输入信号的统计特性的变化自动调整其结构系数，以满足某种最佳准则的要求。下面介绍如何用 Simulink 对 LMS 自适应滤波器进行仿真。

LMS 自适应滤波器仿真结构模型如图 9-25 所示。

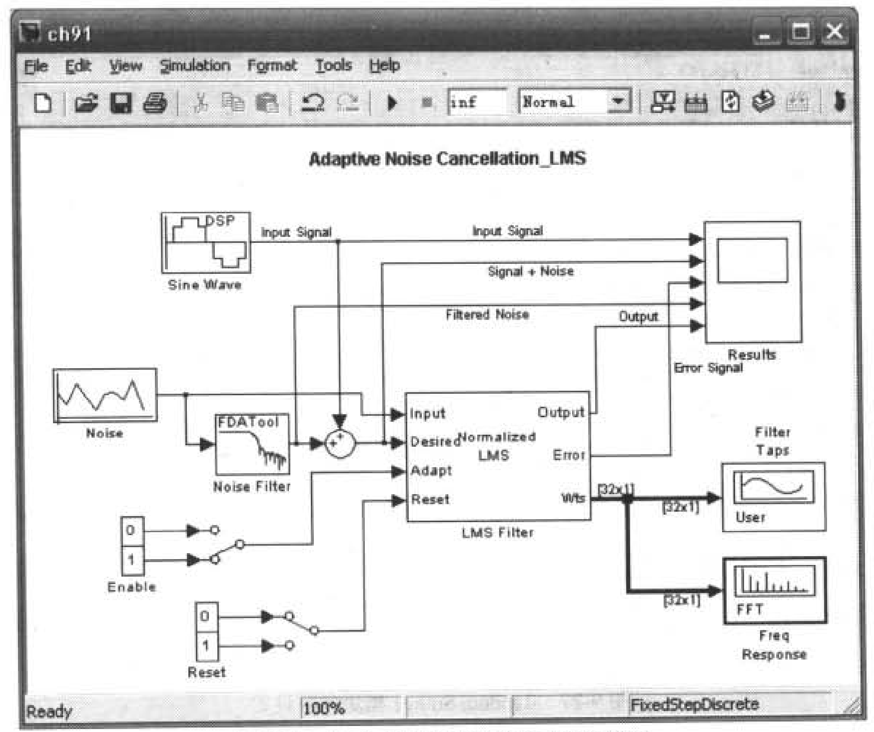


图 9-25　LMS 自适应滤波器仿真结构模型

该模型中，信源由 Sine Wave 模块产生的正弦信号和 Noise 模块产生的白噪声组成，Noise 模块产生均值为零、方差为 1 的 Gaussian 白噪声，作为参考信号输入 LMS 自适应滤波器的 Input 端口，白噪声经过由 Noise Filter 模块设计的滤波器后与正弦信号叠加，作为期望输出信号输入到 LMS 滤波器的 Desired 端口，经过 LMS 滤波器后输出滤波器的系数，通过 Vector Scope 模块显示其时域系数，通过 Spectrum Scope 模块显示其周期图。下面列出各个模块的参数设置：

（1）Sine Wave 模块参数设置如图 9-26 所示。

Main | Data Types

Amplitude:
1

Frequency (Hz):
0.055

Phase offset (rad):
0

Sample mode: Discrete

Output complexity: Real

Computation method: Trigonometric fcn

Sample time:
1

Samples per frame:
1

Resetting states when re-enabled: Restart at time zero

图 9-26　Sine Wave 模块参数设置

数据类型选为 double。

（2）Random Source 模块参数设置如图 9-27 所示。

Parameters

Source type: Gaussian

Method: Ziggurat

Mean:

0

Variance:

1

Repeatability: Specify seed

Initial seed:

[23341]

☐ Inherit output port attributes

Sample mode: Discrete

Sample time:

1

Samples per frame:

1

Output data type: Double

Complexity: Real

图 9-27 Random Source 模块参数设置

（3）Digital Filter Design 模块参数设置如图 9-28 所示。

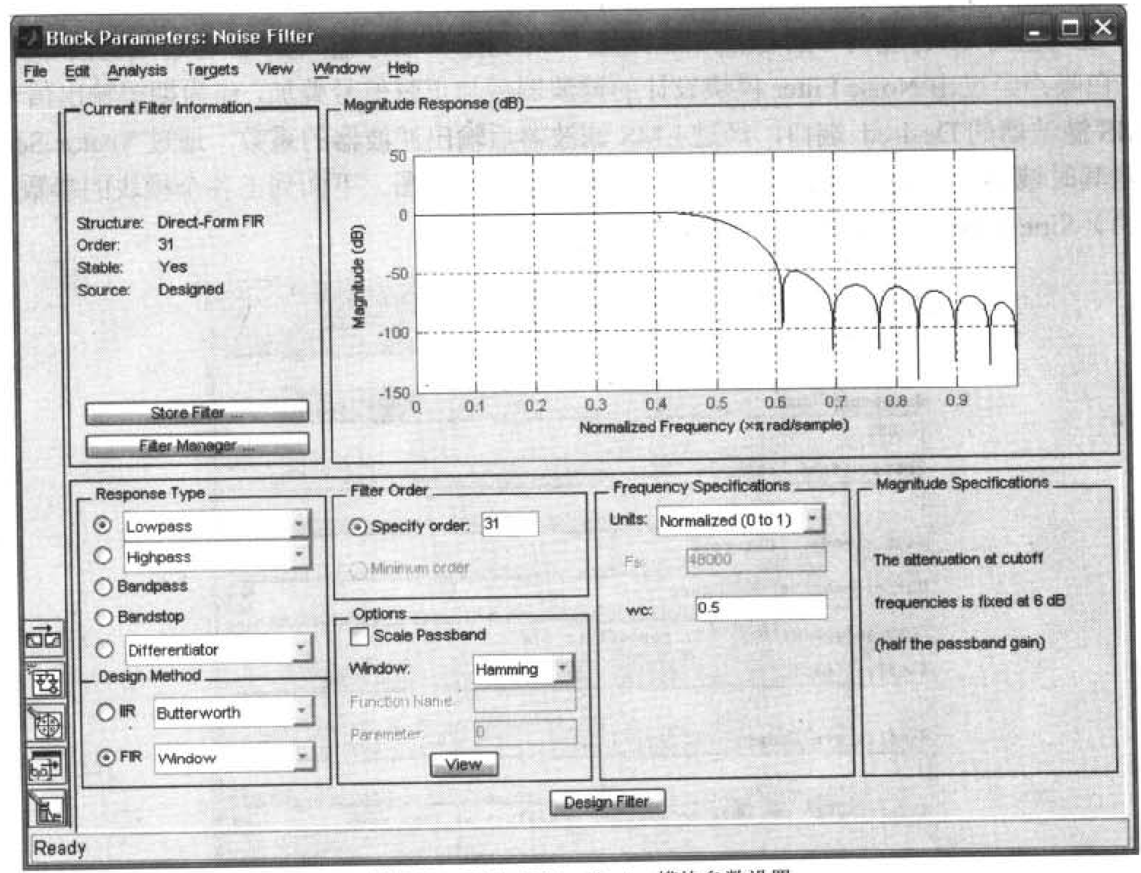


图 9-28 Digital Filter Design 模块参数设置

（4）LMS Filter 模块参数设置如图 9-29 所示。

Main | Fixed-point
Parameters
Algorithm: Normalized LMS
Filter length: 32
Specify step size via: Dialog
Step size (mu): 0.1
Leakage factor (0 to 1): 1.0
Initial value of filter weights: 0
☑ Adapt port
Reset port: Either edge
☑ Output filter weights

图 9-29　LMS Filter 模块参数设置

此例未涉及定点数，故定点数设置采用默认值。

（5）Vector Scope 模块参数设置如下。

➢ Scope Properties 标签设置如图 9-30 所示。

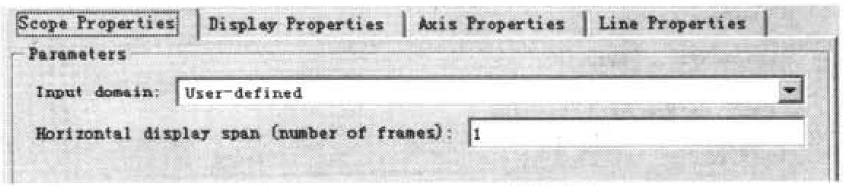


图 9-30　Scope Properties 标签设置

➢ Display Properties 标签设置如图 9-31 所示。

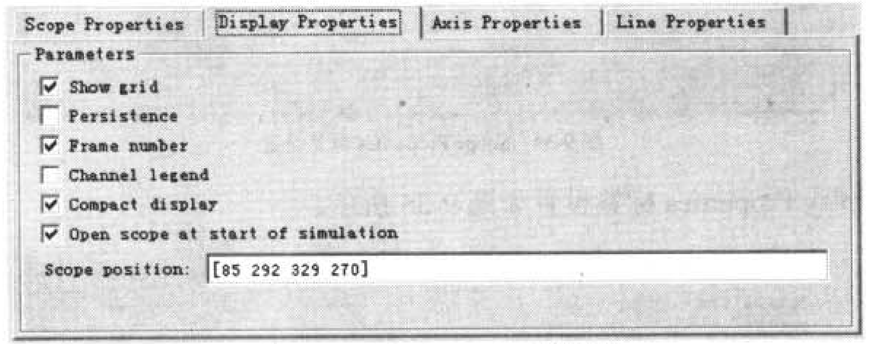


图 9-31　Display Properties 标签设置

➢ Axis Properties 标签设置如图 9-32 所示。

Scope Properties | Display Properties | Axis Properties | Line Properties
Parameters
☑ Inherit sample increment from input
Increment per sample in input frame: 1
X-axis title: Samples
Minimum Y-limit: -0.2
Maximum Y-limit: 0.6
Y-axis title: Filter Coefficients

图 9-32　Axis Properties 标签设置

➢ Line Properties 标签设置如图 9-33 所示。

Scope Properties | Display Properties | Axis Properties | Line Properties
Parameters
Line visibilities: on
Line styles: :
Line markers: .
Line colors: [0 0 1]

图 9-33　Line Properties 标签设置

（6）Spectrum Scope 模块参数设置如下。

➢ Scope Properties 标签设置如图 9-34 所示。

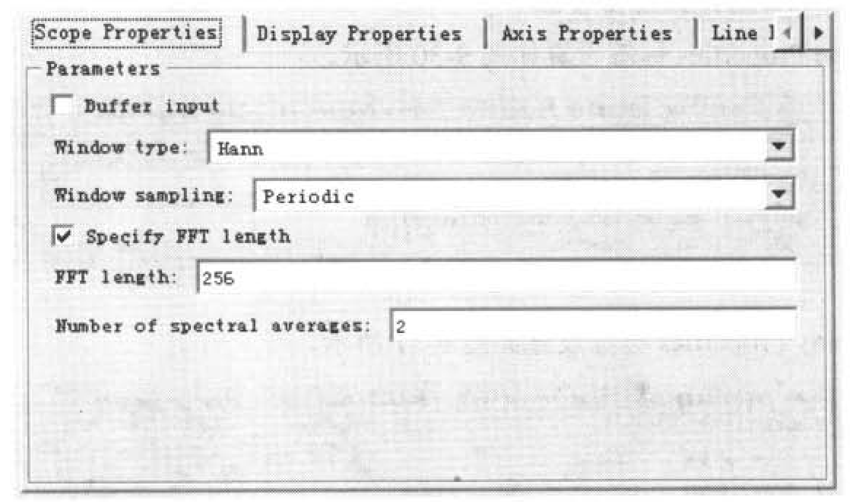


图 9-34　Scope Properties 标签设置

➢ Display Properties 标签设置如图 9-35 所示。

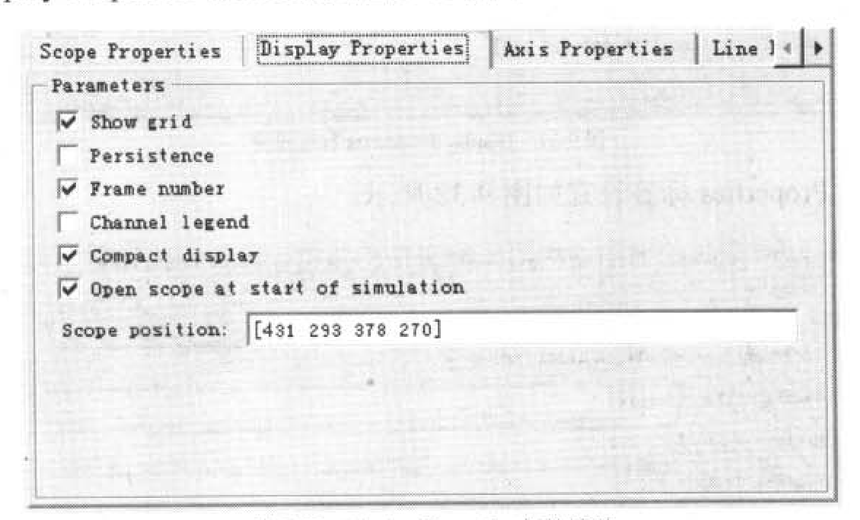


图 9-35　Display Properties 标签设置

➢ Axis Properties 标签设置如图 9-36 所示。

Scope Properties | Display Properties | Axis Properties | Line

Parameters

Frequency units:	Hertz
Frequency range:	[0...Fs/2]
☐ Inherit sample increment from input	
Sample time of original time series:	1
Amplitude scaling:	dB
Minimum Y-limit:	-60
Maximum Y-limit:	0
Y-axis title:	Magnitude-squared, dB

图 9-36　Axis Properties 标签设置

➢ Line Properties 标签设置如图 9-37 所示。

ties | Display Properties | Axis Properties | Line Properties

Parameters

Line visibilities:	on
Line styles:	-
Line markers:	
Line colors:	[0 0 1]

图 9-37　Line Properties 标签设置

运行仿真，刚开始处于训练阶段，各个显示模块如图 9-38、图 9-39、图 9-40 所示，由图可见训练阶段输出尚未达到要求，输出信号与期望输出差别很大。

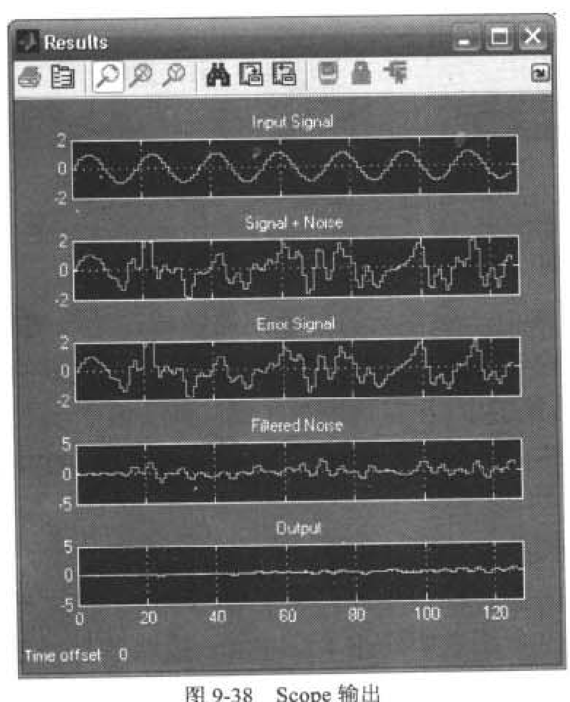



图 9-38　Scope 输出

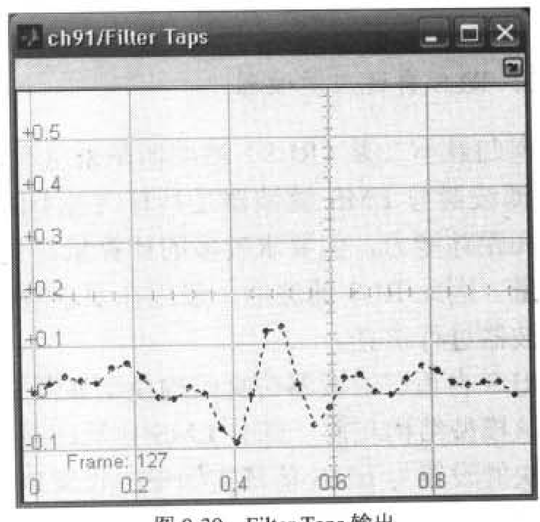


图 9-39　Filter Taps 输出

当训练结束后，如图 9-41、图 9-42、图 9-43 所示，输出信号与期望输出非常接近。

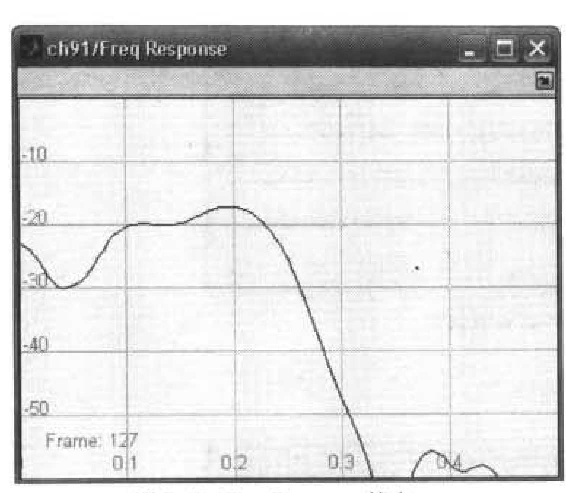


图 9-40　Freq Response 输出

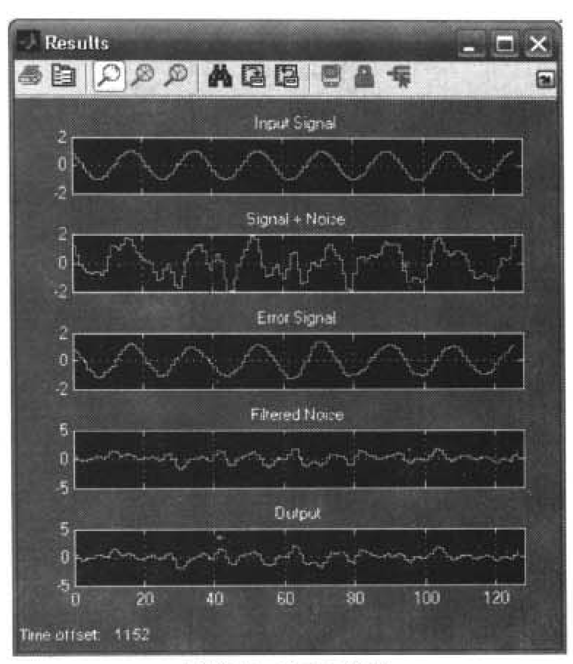


图 9-41　Scope 输出

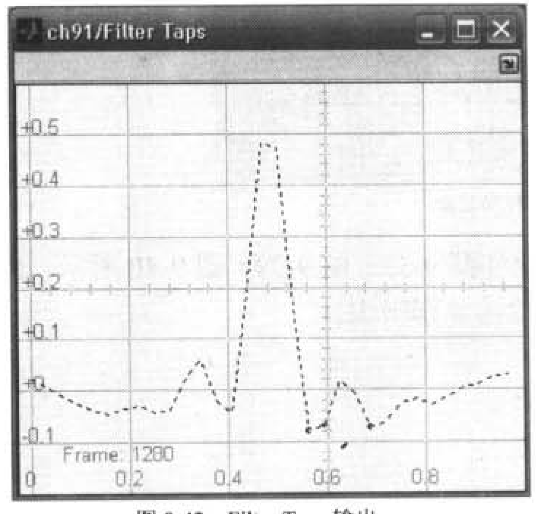


图 9-42　Filter Taps 输出

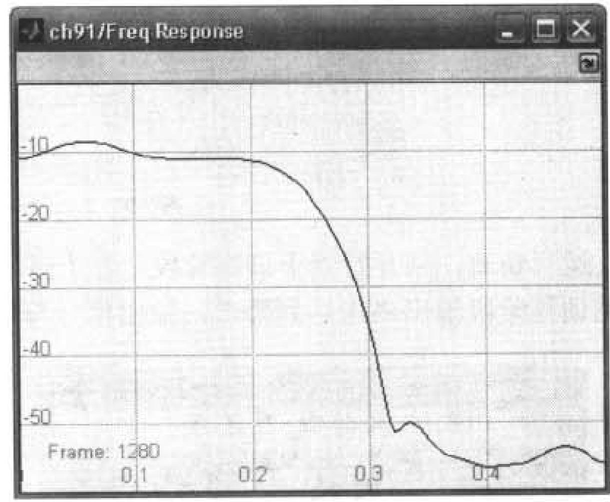


图 9-43　Freq Response 输出

2. RLS 自适应滤波器

递归最小二乘（RLS）滤波器是采用递归的最小二乘算法的自适应滤波器，一般情况下，RLS 滤波器与 LMS 滤波器是具有两种不同工作准则的自适应滤波器，前者具有较好的收敛特性和跟踪能力，但要求较多的计算量，目前最快的 RLS 算法要比 LMS 算法多 2 到 3 倍的计算量。因此 RLS 滤波器一般用在要求较高的场合。下面介绍如何用 Simulink 对 RLS 自适应滤波器进行仿真。

RLS 自适应滤波器仿真结构模型如图 9-44 所示。

该模型结构与前一部分 LMS 自适应滤波器的仿真模型相同，除了 RLS Filter 模块外，其他模块的设置与 LMS 仿真实例中的设置也相同。下面主要给出 RLS Filter 模块的参数设置，如图 9-45 所示。

运行仿真，刚开始处于训练阶段，各个显示模块如图 9-46、图 9-47、图 9-48 所示。

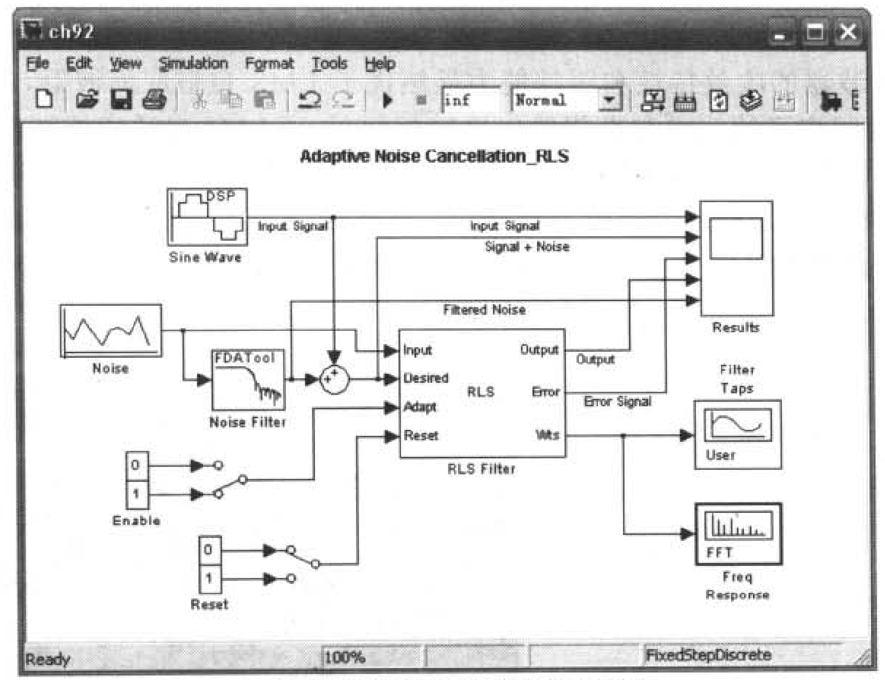


图 9-44　RLS 自适应滤波器仿真结构模型

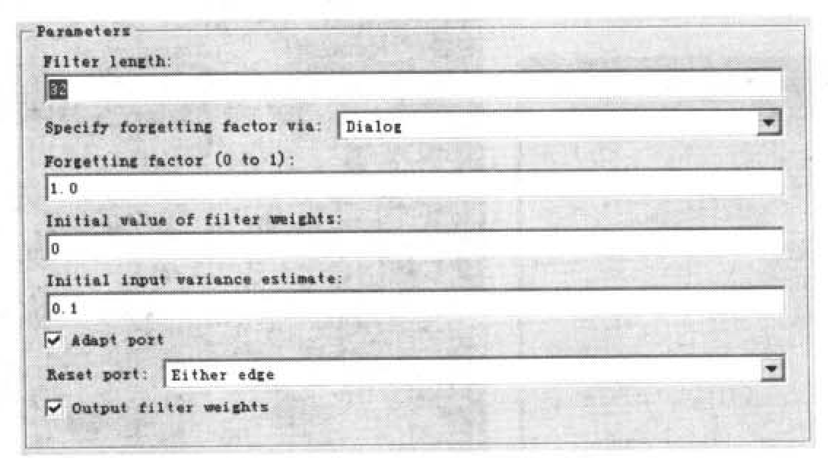


图 9-45　RLS Filter 模块的参数设置

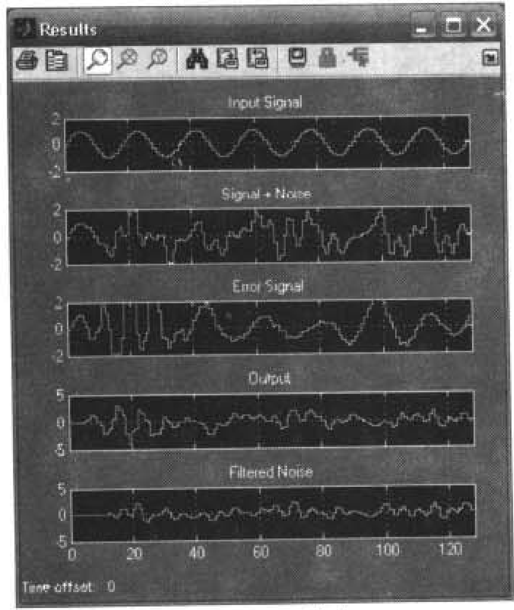


图 9-46　Scope 输出

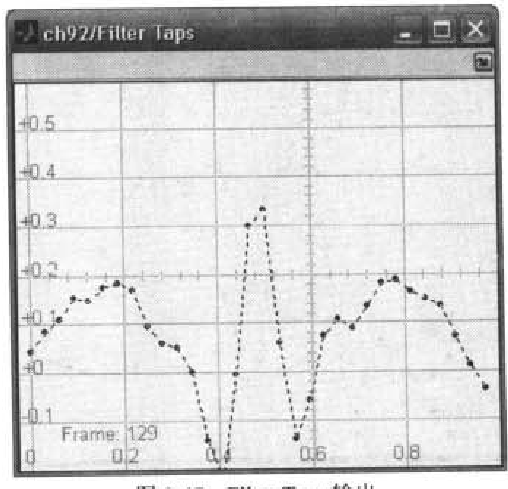


图 9-47　Filter Taps 输出

由图可见，训练阶段输出与期望输出差别较大，但比较此例的 Scope 模块，会发现 RLS 自适应滤波器的收敛特性和跟踪能力显然优于 LMS 自适应滤波器，从刚开始仿真的 40 个仿真时间步开始已经与期望输出比较相似了，而 LMS 自适应滤波器大概要到 100 个仿真时间步以后才开始接近期望输出信号，但接近的速度没有 RLS 算法快，大概要到 600 仿真时间步以后才很好的逼近期望输出，而 RLS 自适应滤波器大概在 400 个时间步以后就很接近期望输出信号了，如图 9-49、图 9-50、图 9-51 所示。这也验证了本实例开始时的论断。

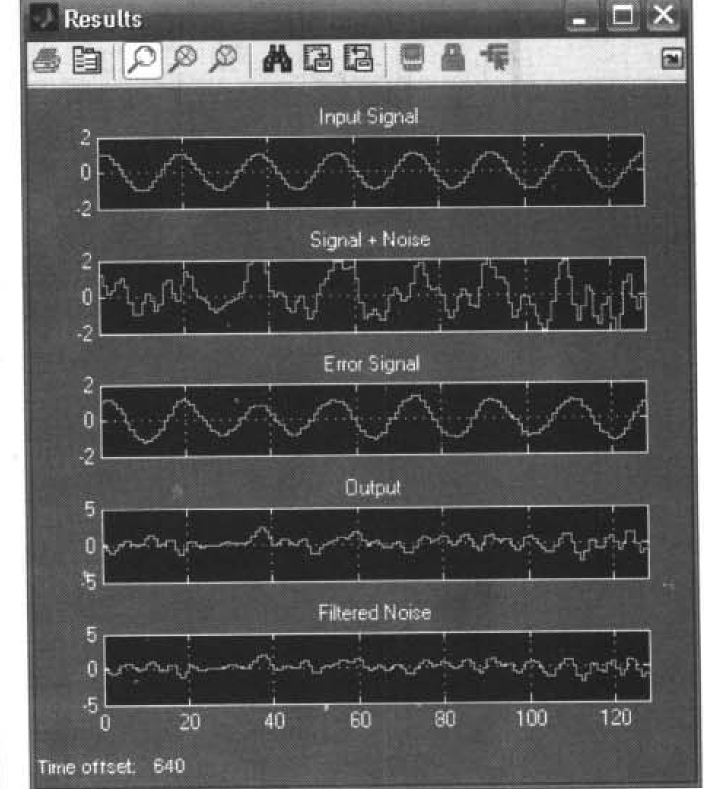


图 9-48 Freq Response 输出


图 9-49 Scope 输出

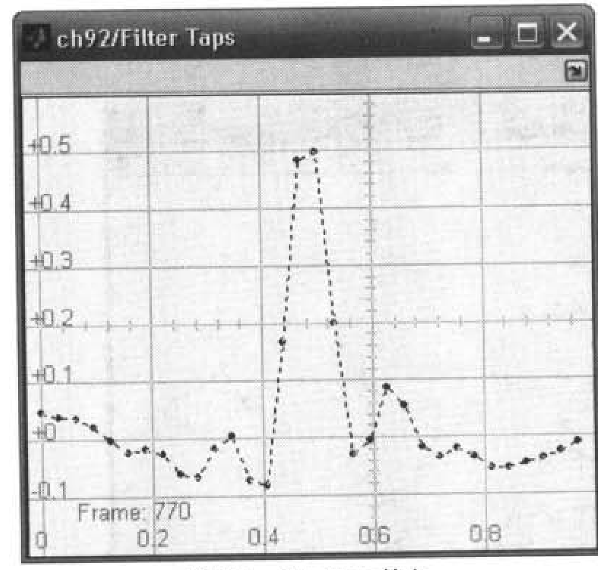


图 9-50 Filter Taps 输出

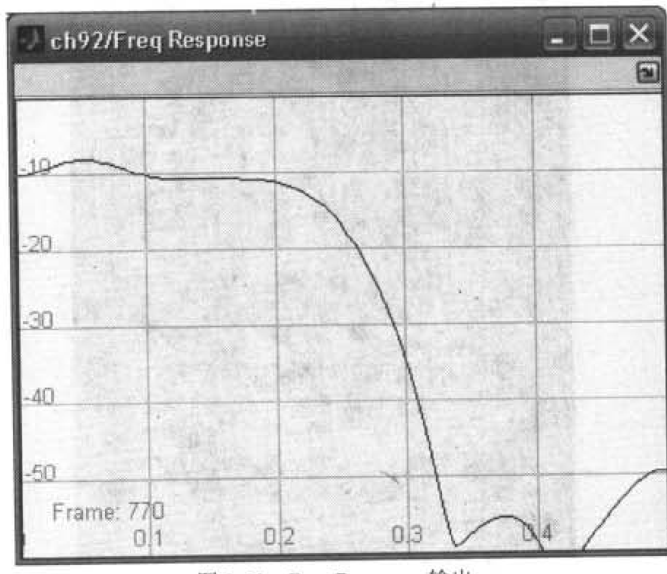


图 9-51 Freq Response 输出

9.4 信号处理仿真实例 2——卡尔曼滤波

卡尔曼滤波器是用前一个估计值和最近一个观察数据（不需要全部过去的观察数据）来估计信号的值，它是用状态方程和递推的方法进行估计，其解是以估计值（常常是状态变量值）形式给出的，因此更常称这种系统为线性最优估计器或滤波器。而且它用递推法计算，不需要知道全部的过去数据，从而运用计算机计算方便，而且它可用于平稳和不平稳的随机过程（信号），非时变和时变的系统。

本节介绍一个利用卡尔曼滤波器来追踪一个 5 阶 FIR 不确定的滤波器系数的实例，从中学习卡尔曼滤波器的应用。仿真模型如图 9-52 所示。

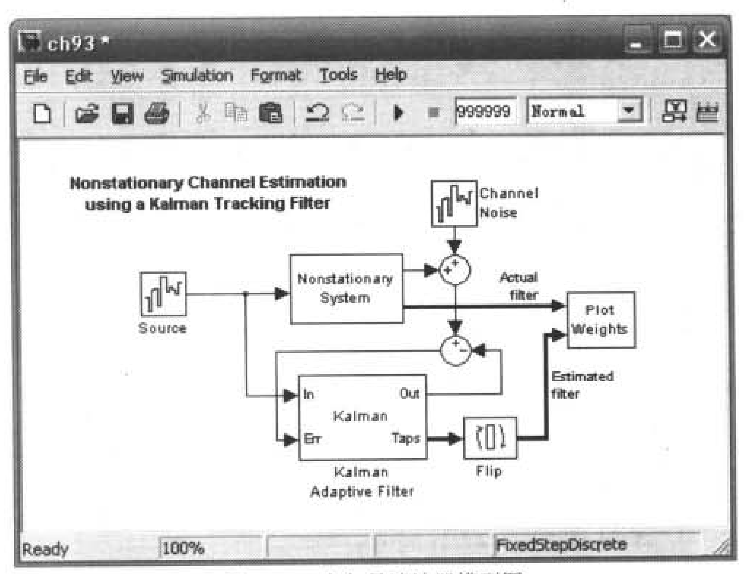


图 9-52　卡尔曼滤波器模型图

该模型中，首先产生窄带白噪声，输入通过一个 5 阶不定 FIR 系统，产生的信号再叠加信道白噪声，叠加后信号与 Kalman 滤波器输出信号作差，降误差输入 Kalman 滤波器，同时窄带白噪声也输入 Kalman 滤波器，Kalman 滤波器根据此误差和输入的原始信号（窄带白噪声）估计 5 阶 FIR 滤波器的系数，估计系数与实际系数同时输入 Plot Weights 子系统，如图 9-53 所示，对比显示输出。

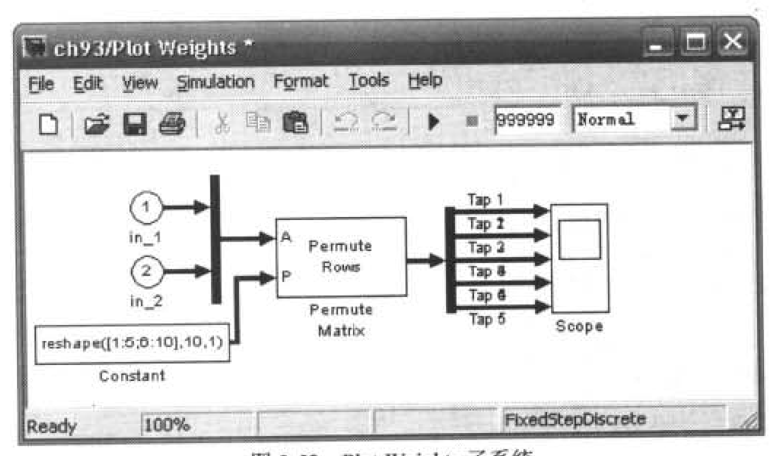


图 9-53　Plot Weights 子系统

下面给出各个模块参数设置：

- Source 模块参数设置如图 9-54 所示。

Parameters
Noise power:
[0.1]
Sample time:
1
Seed:
[23341]
Interpret vector parameters as 1-D

图 9-54　Source 模块参数设置

- Nonstationary system 模块参数设置如图 9-55 所示。

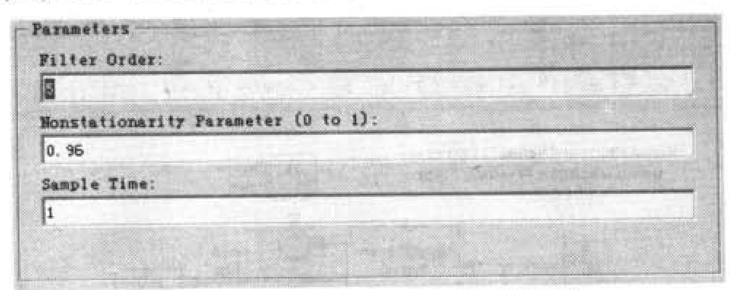


图 9-55　Nonstationary system 模块参数设置

- Channel Noise 模块参数设置如图 9-56 所示。

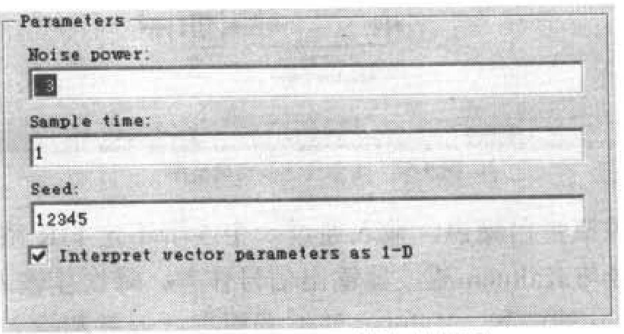


图 9-56　Channel Noise 模块参数设置

- Kalman 模块参数设置如图 9-57 所示。

Parameters
FIR filter length:
8
Measurement noise variance:
0.3
Process noise variance:
0.1
Initial value of filter taps:
0
Initial error correlation matrix:
0.5
Adapt port
Reset port: None

图 9-57　Kalman 模块参数设置

- Flip 模块参数设置如图 9-58 所示。

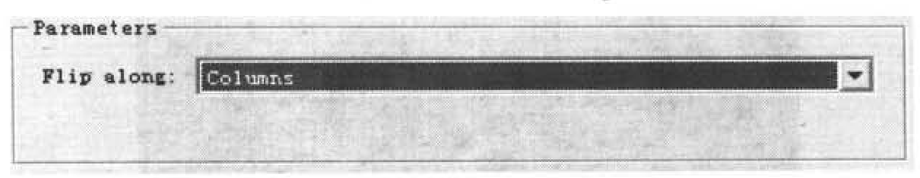


图 9-58　Flip 模块参数设置

- Constant 模块参数设置如下。
  - ➢ Main 标签参数设置如图 9-59 所示。

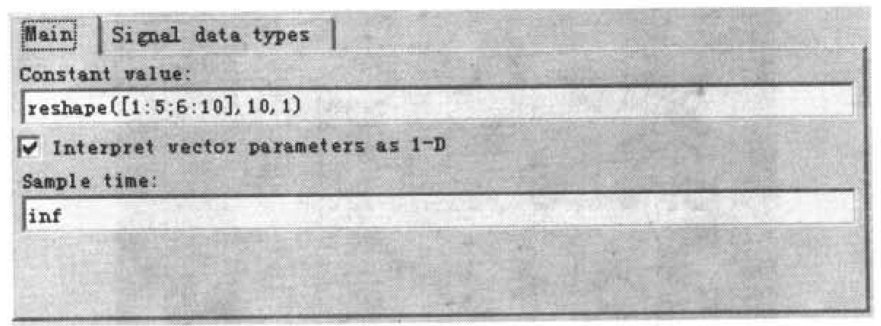


图 9-59　Main 标签参数设置

  - ➢ Signal data types 标签参数设置如图 9-60 所示。

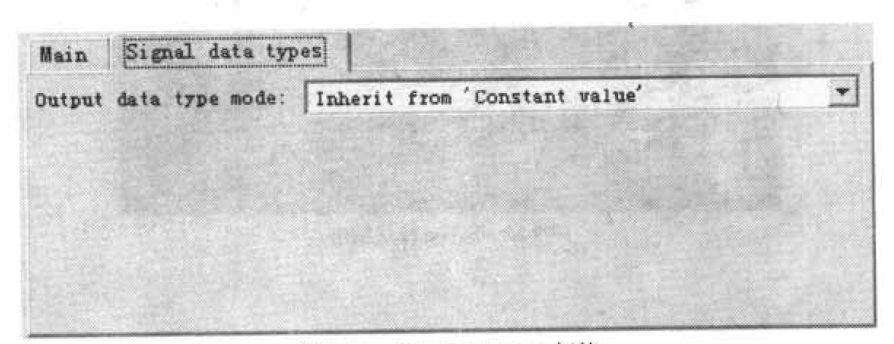


图 9-60　Signal data types 标签

- Permute Rows 模块参数设置如图 9-61 所示。

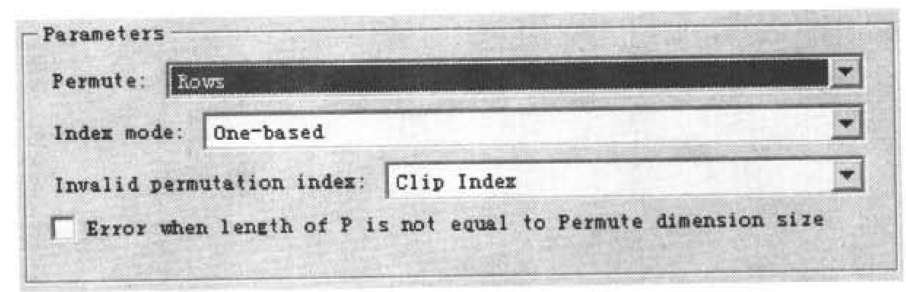


图 9-61　Permute Rows 模块参数设置

运行仿真输出观察 Scope 模块输出如图 9-62 所示。

波形稍微靠后一点的为 Kalman 滤波器输出的估计系数，可见卡尔曼滤波器对非平稳信号具有较好的预测效果。

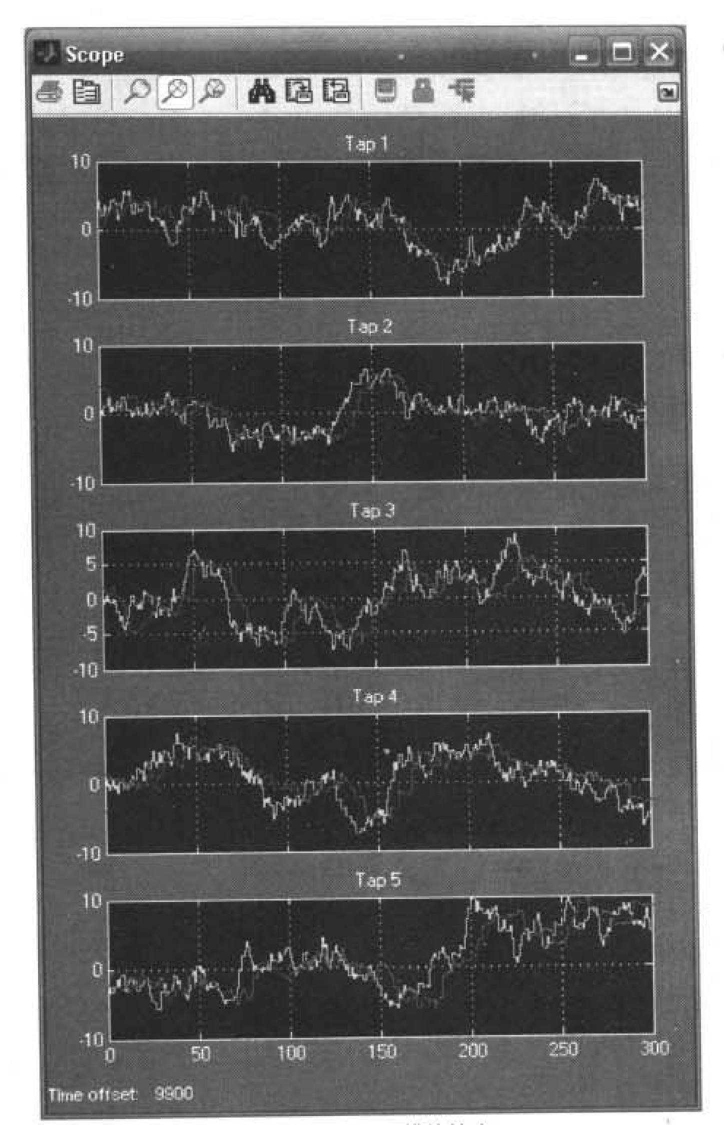


图 9-62 Scope 模块输出

第 10 章　Simulink6.0 在通信系统仿真中的应用

本章主要介绍通信系统仿真的基础知识及运用 Simulink6.0 进行仿真的方法，介绍 Simulink6.0 中常用的通信仿真模块，然后通过实例说明如何把这些知识应用与实践。

本章主要内容：

- 通信系统仿真基础
- Simulink6.0 中通信系统仿真模块
- 通信系统仿真实例 1——数字幅度调制的抗噪声性能
- 通信系统仿真实例 2——QPSK 与 DQPSK 性能之比较

10.1　通信系统仿真基础

仿真是衡量系统性能的工具，它通过仿真模型的仿真结果来推断原系统的性能，从而为新系统的建立或原系统的改造提供可靠的参考。通过仿真，可以降低新系统失败的可能性，消除系统中潜在的瓶颈，防止对系统中某些功能部件造成过量的负载，优化系统的整体性能，因此，仿真是科学研究和工程建设中不可缺少的方法。

实际的通信系统是一个功能结构相当复杂的系统，对这个系统做出的任何改变（如改变某个参数的设置、改变系统的结构等）都可能影响到整个系统的性能和稳定。因此，在对原有的通信系统做出改进或建立一个新系统之前，通常需要对这个系统进行建模和仿真，通过仿真结果衡量方案的可行性，从中选择最合理的系统配置和参数设置，然后再应用于实际系统中。这个过程就是通信仿真。

10.1.1　通信系统仿真简介

通信仿真是衡量通信系统性能的工具。通信仿真可以分成离散事件仿真和连续仿真。在离散事件仿真中，仿真系统只对离散事件做出响应，而在连续仿真中，仿真系统对输入信号产生连续的输出信号；离散事件仿真是对实际通信系统的一种简化，它的仿真建模比较简单，整个仿真过程需要花费的时间也比连续仿真少。虽然离散事件仿真舍弃了一些仿真细节，在有些场合显得不够具体，但仍然是通信仿真的主要形式。

与一般的仿真过程类似，在对通信系统实施仿真之前，首先需要研究通信系统的特性，通过归纳和抽象建立通信系统的仿真模型。图 10-1 所示是关于通信系统仿真流程的一个示意图。从图中可以看到，通信系统仿真是一个循环往复的过程，它从当前系统出发，通过分析建立起一个能够在一定程度上描述原通信系统的仿真模型，然后通过仿真实验得到相关的数

据，通过对仿真数据的分析可以得到相应的结论，然后把这个结论应用到对当前通信系统的改造中，如果改造后通信系统的性能并不像仿真结果那样令人满意，还需要重新实施通信系统仿真，这时候改造后的通信系统就成了当前系统，并且开始新一轮的通信系统仿真过程。

值得注意的是，在整个通信系统的仿真过程中，人为因素自始至终起着相当重要的作用。除了仿真程序的运行之外，通信仿真的每个步骤都需要进行人工干预，由人对当前的情况做出正确的判断。因此，通信仿真并不是一个机械的过程，它实际上是人的思维活动在计算机协助下的一种延伸。

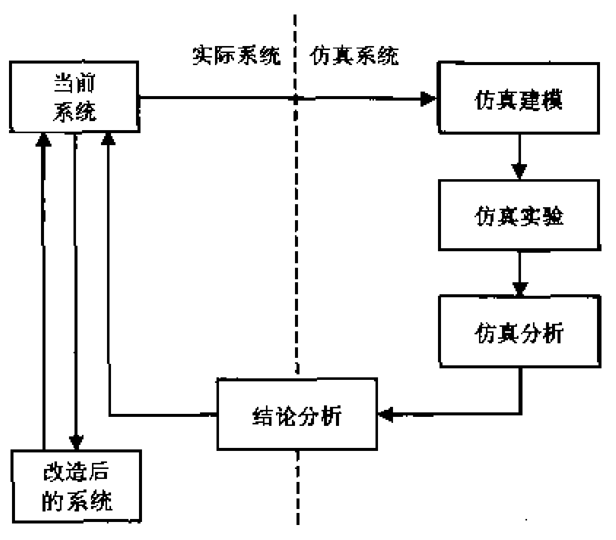


图 10-1　通信系统仿真流程的一个示意图

10.1.2　通信系统仿真流程

通信系统仿真一般分成 3 个步骤，即仿真建模、仿真实验和仿真分析。应该注意的是，通信仿真是一个螺旋式发展的过程，因此这 3 个步骤可能需要循环执行多次之后才能够获得令人满意的仿真结果。

1. 仿真建模

仿真建模是根据实际通信系统建立仿真模型的过程，它是整个通信仿真过程中的一个关键步骤，因为仿真模型的好坏直接影响着仿真的结果以及仿真结果的真实性和可靠性。

仿真模型是对实际系统的一种模拟和抽象，但又不是完全的复制。简单的仿真模型容易被理解和操作，但是由于它忽略了很多关于实际系统的细节，因而在一定程度上影响了仿真的可靠性。如果仿真模型比较复杂，虽然它是对实际系统的一种忠实反映，但是其中包含了过多的相互作用因素，这些因素不仅需要消耗过多的仿真时间，而且使仿真结果的分析过程变得相当复杂。因此，仿真模型的建立需要综合考虑其可行性和简单性。在仿真建模过程中，我们可以先建立一个相对简单的仿真模型，然后再根据仿真结果和仿真过程的需要逐步增加仿真模型的复杂度。

仿真模型一般是一个数学模型。数学模型有多种分类方式，包括确定性（Deterministic）模型和随机（Statistic）模型，静态（Static）模型和动态模型（Dynamic）。确定性模型的输入变量和输出变量都有固定数值，而在随机模型中，至少有一个输入变量是随机的。静态模型不需要考虑时间变化因素，动态模型的输入输出变量需要考虑时间变化因素。一般情况下通信仿真模型是一个随机动态系统。

在仿真建模过程中，首先需要分析实际系统存在的问题或设立系统改造的目标，并且把这些问题和目标转化成数学变量和公式。例如，我们可以设定改造后系统或新系统在达到系统最大容量时的误帧率，或者是通信系统的最大呼损率，等等。

有了这些具体的仿真目标之后，下一步是获取实际通信系统的各种运行参数，如通信系统占用的带宽及其频率分布，系统对于特定的输入信号产生的输出等。同时，对于通信系统

中的各个随机变量，可以采集这些变量的数据，然后通过数学工具来确定随机变量的分布特性。

有了上面的准备工作，下一步就可以通过仿真软件来建造仿真模型了。最简单的工具是采用 C 语言等编程工具直接编写仿真程序，这种方法的优点是效率高，缺点则是不够灵活，没有一个易于实现的人机交互界面，不便于对仿真结果进行分析。除此之外，还可以采用专门的仿真软件建造仿真模型，比较常用的仿真软件包括 MATLAB、OPNET、NS2 等，这些软件具有各自不同的特点，适用于不同层次的通信仿真。例如，物理层仿真通常采用 MATLAB．而网络层仿真则适合采用 OPNET。

用仿真软件完成仿真模型之后，还需要对这个仿真模型的有效性进行初步的验证。一种简便的验证方法是采用特定的已知输入信号，这个输入信号分别通过仿真模型和实际系统，产生两种输出信号。如果仿真模型的输出信号与实际系统的输出信号比较吻合，说明这个仿真模型与原系统具有较好的相似性。当这两种输出信号差别很大时，最好先检查一下仿真模型的内部连接和设置，找出造成这种差异的原因。

仿真建模的最后一步是做好仿真模型的文档工作，这是最容易被大家忽略的。很多情况下，我们在完成系统的设计之后就迫不及待地运行仿真程序，等发现仿真结果与预期目标相差甚远时才回过头来焦头烂额地检查仿真模型的内部结构。这时候，往往原先的很多参数设置和条件假设都变得不可理解，这非常不利于修改参数和结构，不利于找错和排错。

2. *仿真实验*

仿真实验是一个或一系列针对仿真模型的测试。在仿真实验过程中，通常需要多次改变仿真模型输入信号的数值，以观察和分析仿真模型对这些输入信号的反应，以及仿真系统在这个过程中表现出来的性能。需要强调的一点是，仿真过程中使用的输入数据必须具有一定的代表性，即能够从各个角度显著地改变仿真输出信号的数值。

实施仿真之前需要确定的另外一个因素是性能尺度。性能尺度指的是能够衡量仿真过程中系统性能的输出信号的数值（或根据输出信号计算得到的数值），因此，在实施仿真之前，首先需要确定仿真过程中应该收集哪些仿真数据，这些数据以什么样的格式存在，以及收集多少数据。

在明确了仿真系统对输入信号和输出信号的要求之后，最好把这些设置整理成一份简单的文档，编写文档是一个好习惯，它能够帮助我们回忆起仿真设计过程的一些细节。当然，文档的编写不一定要求很规范，并且文档大小应视仿真设计的规模而定。

最后，还应该明确各个输入信号的初始设置以及仿真系统内部各个状态的初始值。仿真的运行实际上是计算机的计算过程，这个过程一般情况下需要人工干预，花费的时间由仿真的复杂度确定。如果需要比较仿真系统在不同参数设置下的性能，应该使仿真系统在取不同参数值时具有相同的输入信号（或相同的随机输入信号），这样才能够保证分析和比较的客观性和可靠性。

对于需要较长时间的仿真，应该尽可能地使用批处理方式，使得仿真过程在完成一种参数配置的仿真之后能够自动启动针对下一个参数配置下一个仿真。这种方式减少了仿真过程中的人工干预，提高了系统利用率和仿真效率。

3. *仿真分析*

仿真分析是一个通信仿真过程中的最后一个步骤，在仿真过程中，用户已经从仿真过程中获得了足够多的关于系统性能的信息，但是这些信息只是一些原始数据，一般还需要经过

数值分析和处理才能够获得衡量系统性能的尺度，从而获得对仿真性能的一个总体评价。常用的系统性能尺度包括平均值、方差、标准差、最大值和最小值等，它们从不同的角度描绘了仿真系统的性能。

如果仿真过程需要一定的时间才能够达到平衡状态，在对输出数据进行分析和处理时一般要忽略最初的若干个数据，而只考虑平衡之后的输出。对于仿真尺度不随时间变化的贫困系统（Stationary System），还可能涉及到对输出变量稳定状态的求解。

另外一个需要注意的地方是，即使仿真过程中收集的数据正确无误，由此得到的仿真结果不一定就是准确的，造成这种结果的原因可能是输入信号恰好与仿真系统的内部特性相吻合，或者输入的随机信号不具有足够的代表性。

图表是最简洁的说明工具，它具有很强的直观性，便于分析和比较，因此，仿真分析的结果一般都绘制成图表形式，我们使用的仿真工具一般都具有很强的绘图功能，能够便捷地绘制各种类型的图表。

以上就是通信仿真的一个循环。应该强调的是，仿真分析并不一定意味着通信仿真过程的完全结束。如果仿真分析得到的结果达不到预期的目标，用户还需要重新修改通信仿真模型，这时候仿真分析就成为了另外一个循环的开始。

10.2 Simulink6.0 中通信系统仿真模块

Simulink6.0 提供了丰富的通信系统仿真模块，几乎包括了通信系统仿真中所用到的所有信源、信宿、操作和算法。如图 10-2 所示，用户可以利用这些模块方便地完成自己通信系统的仿真和分析，这一节将分别介绍各个子库。

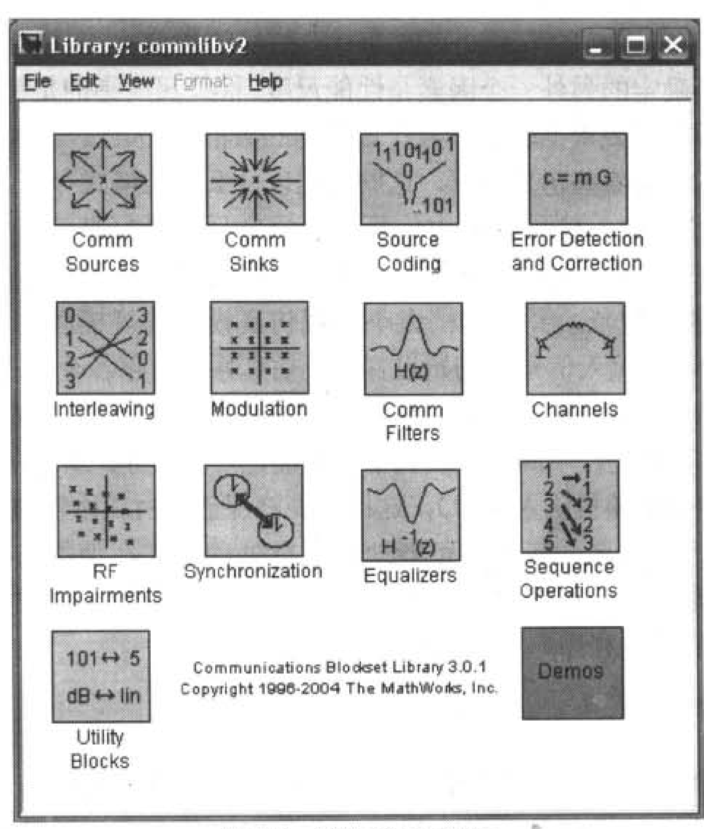


图 10-2 通信系统仿真模块

10.2.1 Comm Sources 子模块集

Comm Sources 子模块集有如下三个大类：Noise Generator 模块组、Random Data Sources 模块组、Sequence Generator 模块，如图 10-3 所示。

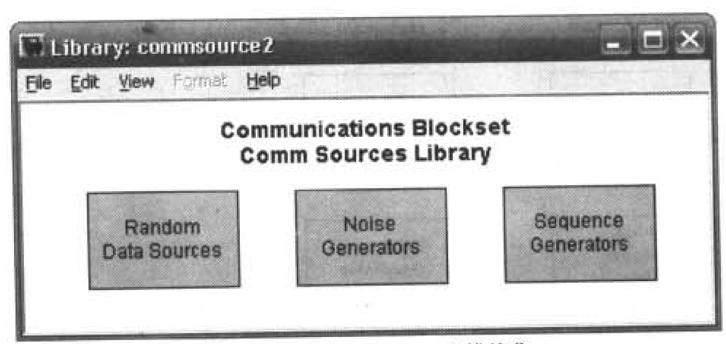


图 10-3 Comm Sources 子模块集

- Noise Generator 模块组包含以下模块，如图 10-4 所示，分别是不同的噪声发生器模块。包括高斯白噪声、均匀噪声、瑞利噪声等。
- Random Data Sources 模块组包含以下模块，如图 10-5 所示，分别是不同的随机数产生模块。包括伯努利分布发生器、泊松分布发生器、均匀随机整数发生器等。

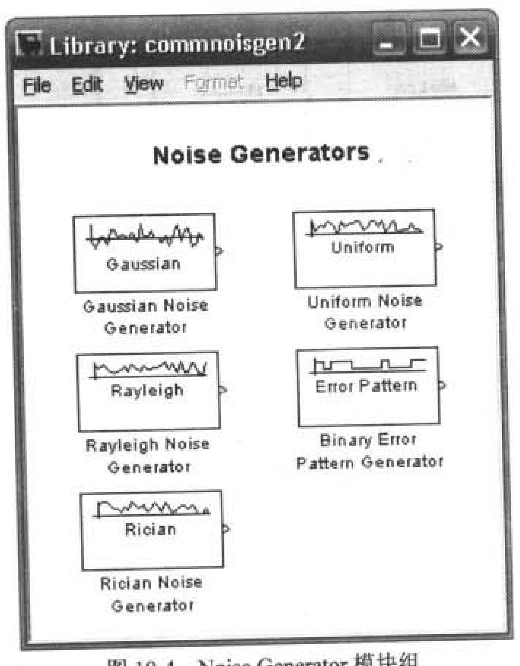


图 10-4 Noise Generator 模块组

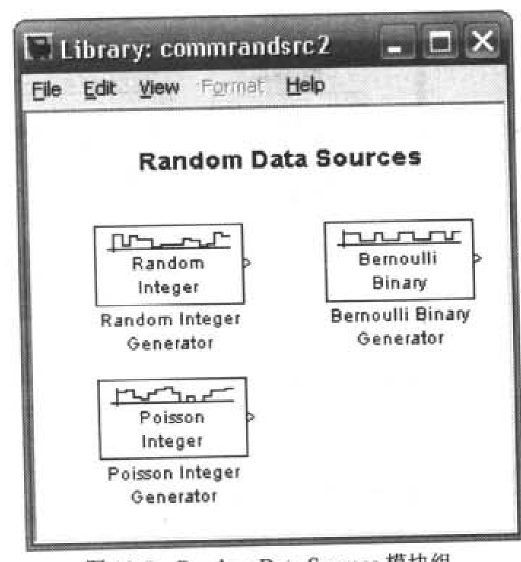


图 10-5 Random Data Sources 模块组

- Sequence Generator 模块包含以下模块，如图 10-6 所示，分别是不同的序列发生器。

10.2.2 Source Coding 子模块集

Source Coding 子模块集主要包括各种信号编码、解码工具，如图 10-7 所示。

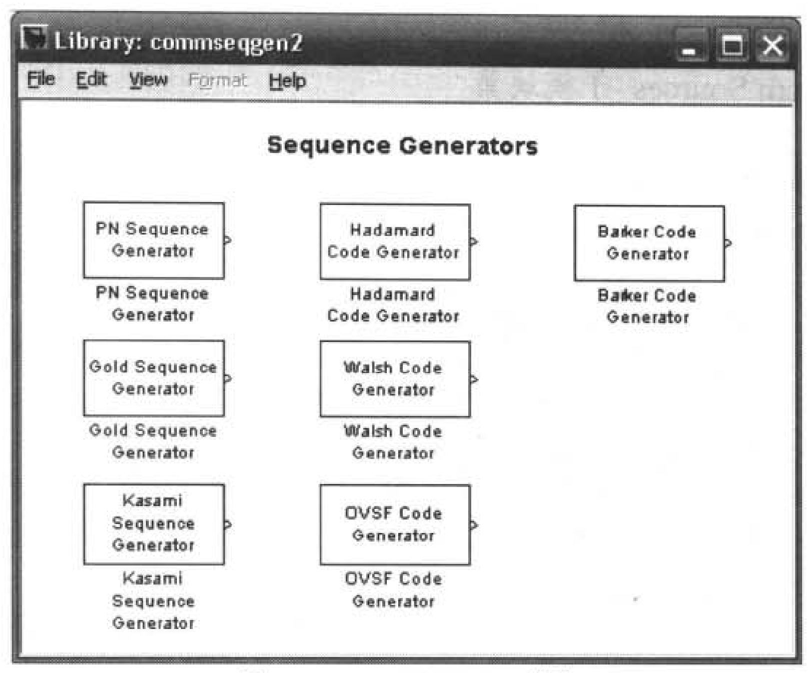


图 10-6 Sequence Generator 模块

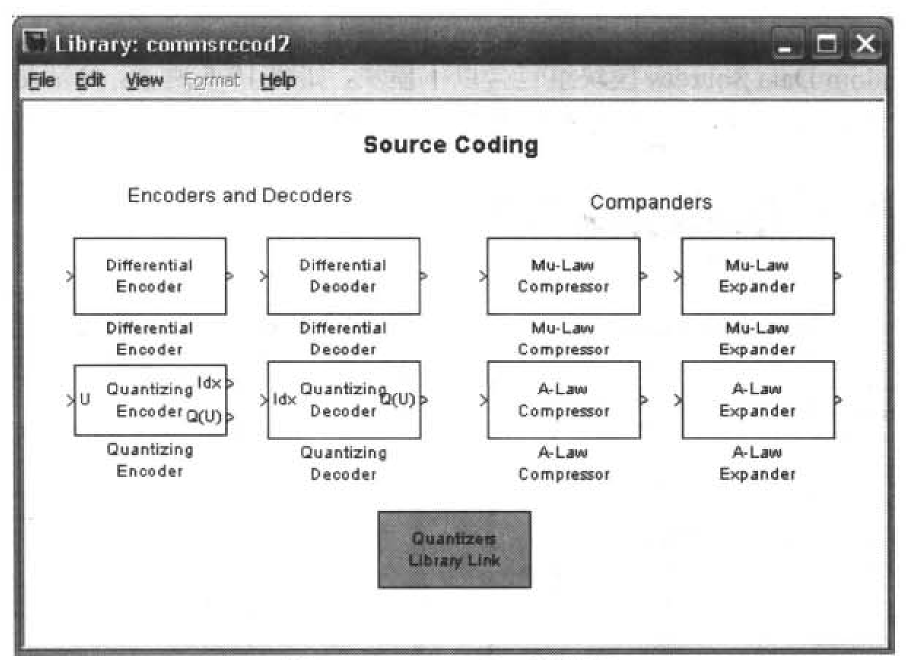


图 10-7 Source Coding 子模块集

10.2.3 Channels 子模块集

Channels 子模块集主要包括各种通道仿真模块，包括引入白噪声的信道、引入二进制错误的信道、引入多径瑞利衰落的信道，如图 10-8 所示。

10.2.4 Comm Sinks 子模块集

Comm Sinks 子模块集包含以下模块，如图 10-9 所示，分别是不同的信宿模块，用来可视化输出接收端的信号。

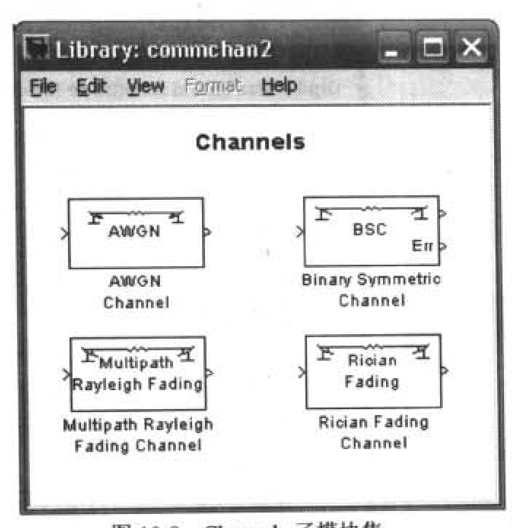


图 10-8　Channels 子模块集

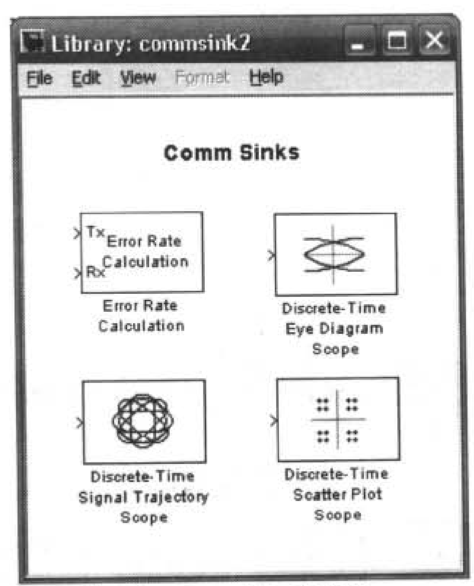


图 10-9　Comm Sinks 模块集

10.2.5　Modulation 子模块集

Modulation 子模块集包括两个大类：Analog Passband Modulation 模块组和 Digital Baseband Modulation 模块组，如图 10-10 所示。

- Analog Passband Modulation 模块组包含以下模块，如图 10-11 所示，分别是不同模拟信号调制、解调模块。
- Digital Baseband Modulation 模块组包含以下模块，如图 10-12 所示，分别是不同数字信号调制、解调模块。

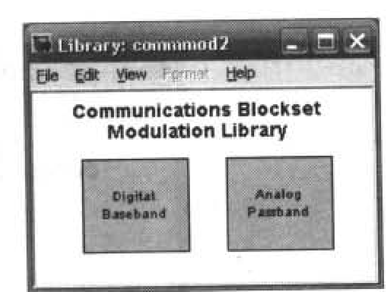


图 10-10　Modulation 子模块集

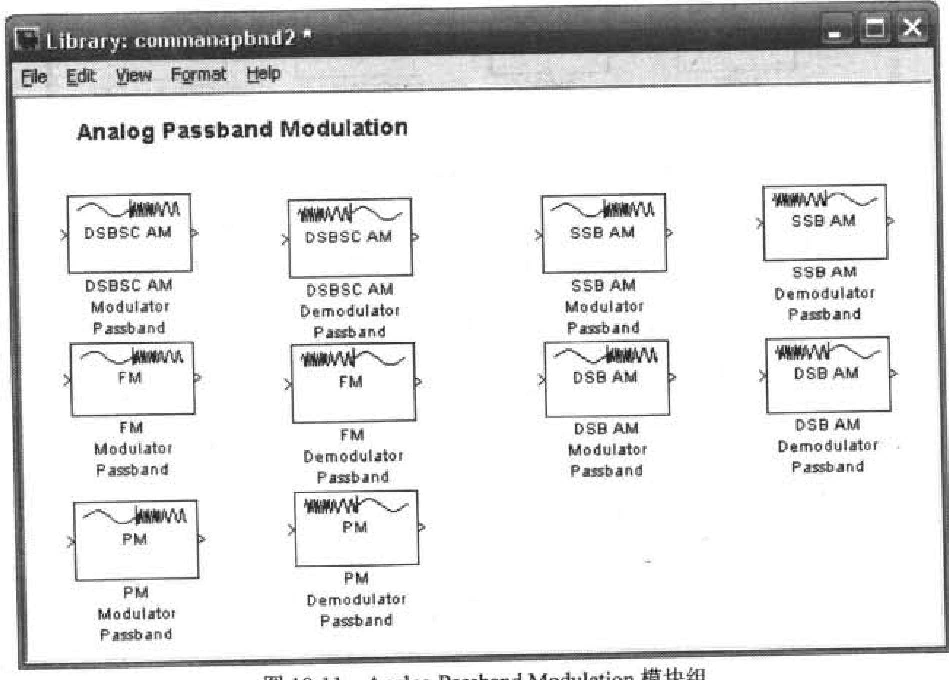


图 10-11　Analog Passband Modulation 模块组

➢ AM 子模块组主要包括各种幅度调制、解调模块，如图 10-13 所示。

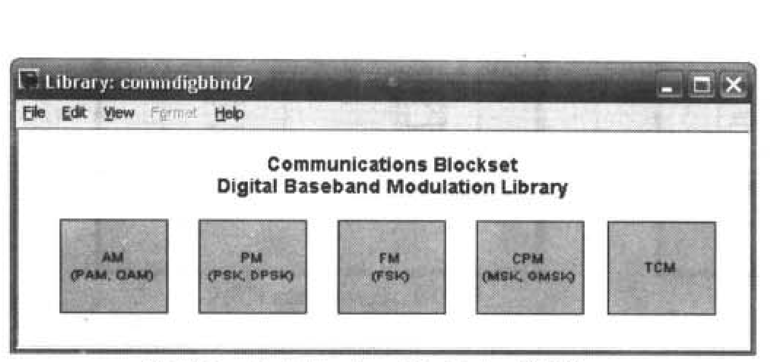




图 10-12 Digital Baseband Modulation 模块组

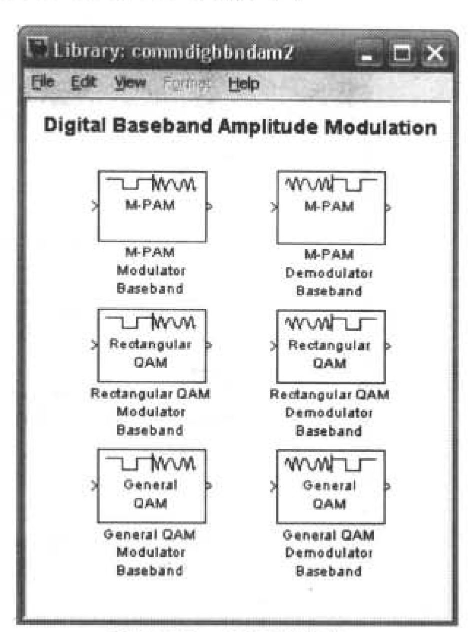


图 10-13 AM 子模块组

➢ CPM 子模块组主要包括各种连续相位调制、解调模块，如图 10-14 所示。

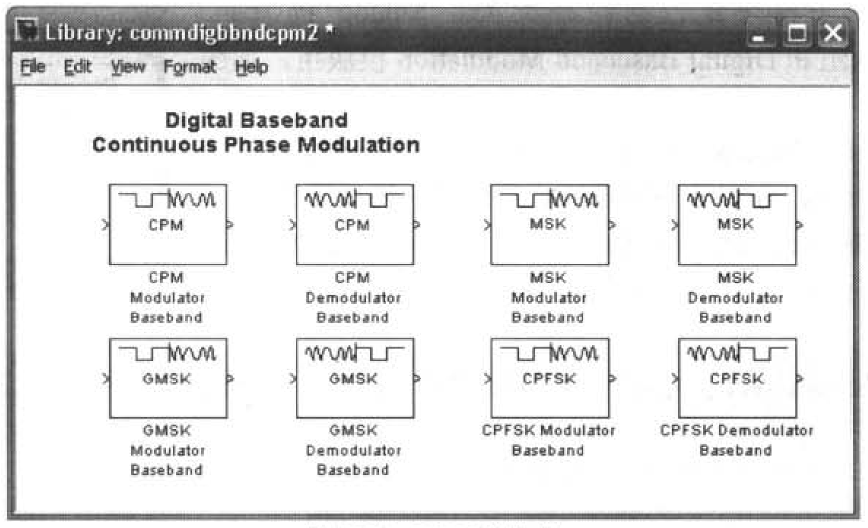


图 10-14 CPM 子模块组

➢ FM 子模块组主要包括各种频率调制解调模块，如图 10-15 所示。

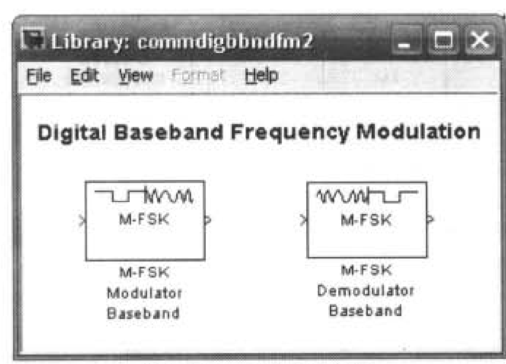


图 10-15 FM 子模块组

➢ PM 子模块组主要包括各种相位调制解调模块，如图 10-16 所示。

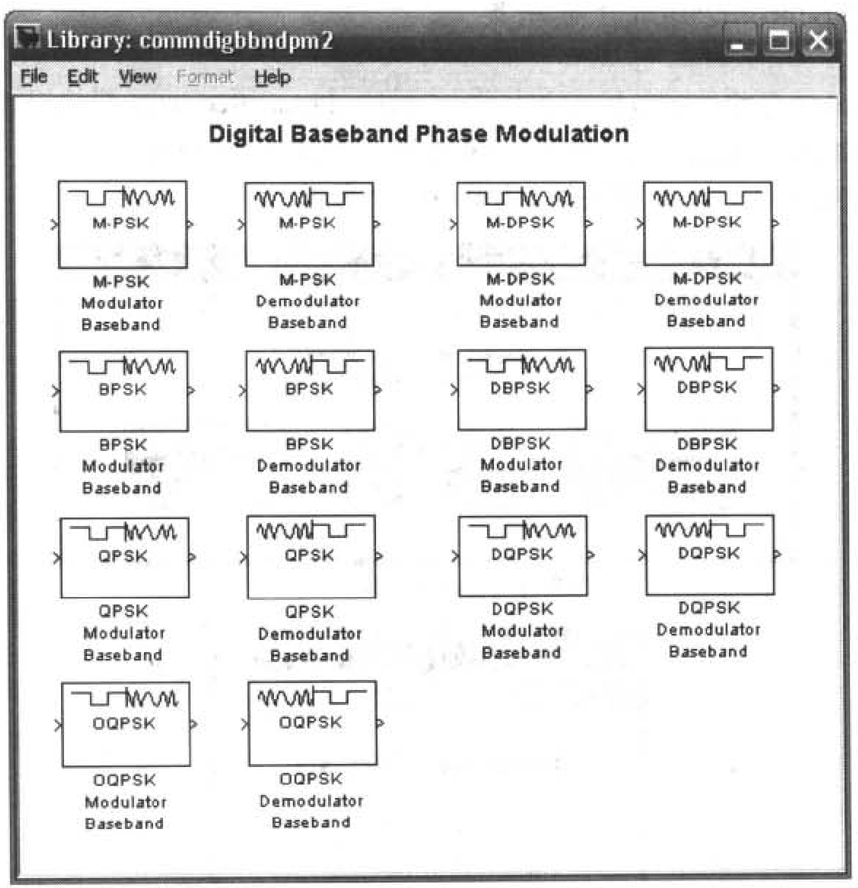


图 10-16 PM 子模块组

➢ TCM 子模块组主要包括各种 TC 调制解调模块，如图 10-17 所示。

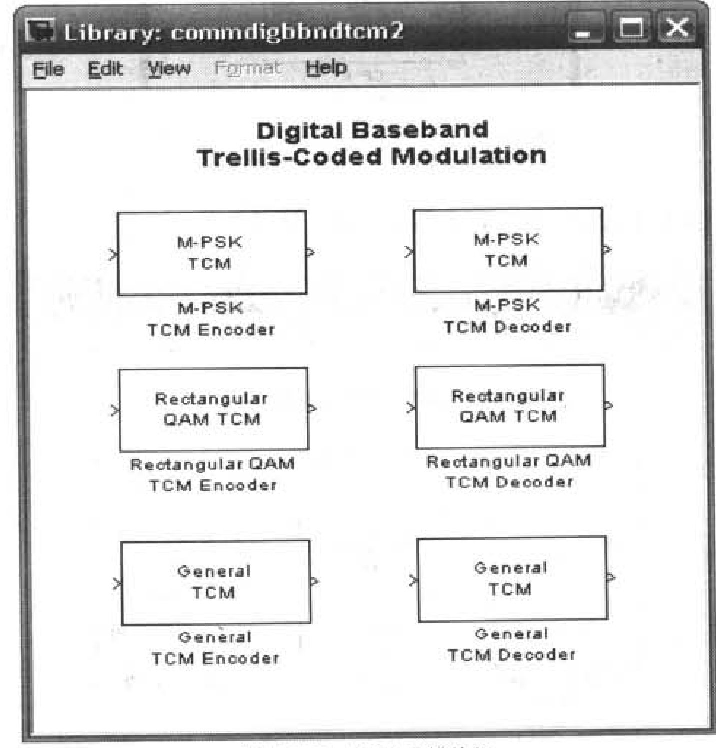


图 10-17 TCM 子模块组

10.2.6 Synchronization 子模块集

Synchronization 子模块集包括三个大类：Carrier Phase Recovery 模块组、Synchronization Components 模块组、Timing Phase Recovery 模块，如图 10-18 所示。

- Carrier Phase Recovery 模块组，如图 10-19 所示。

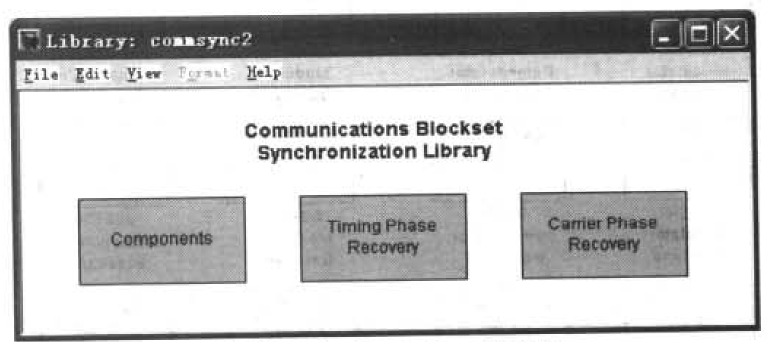


图 10-18 Synchronization 子模块集

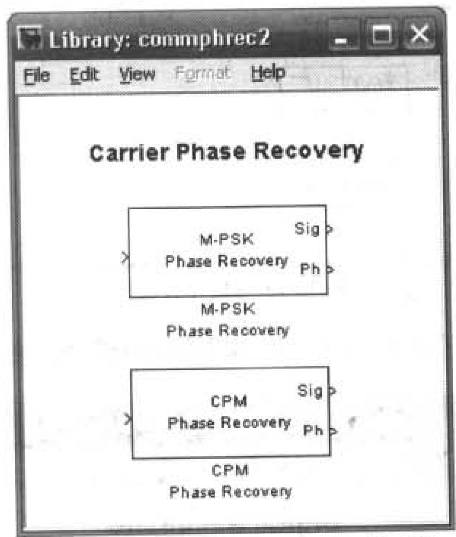


图 10-19 Carrier Phase Recovery 模块组

- Synchronization Components 模块组，如图 10-20 所示。

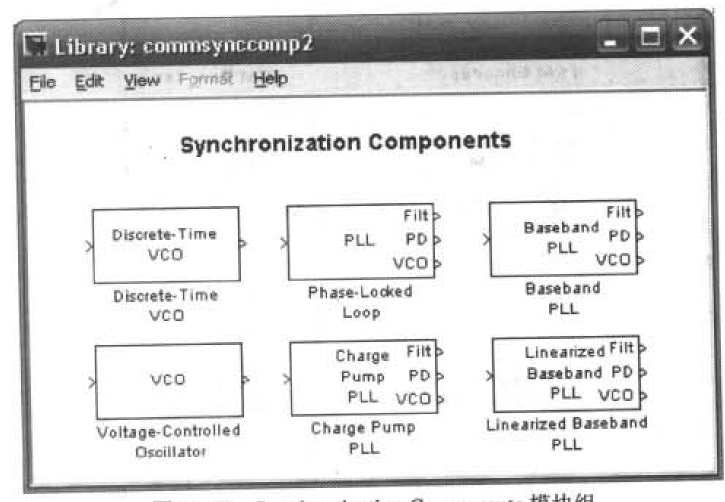


图 10-20 Synchronization Components 模块组

- Timing Phase Recovery 模块组，如图 10-21 所示。

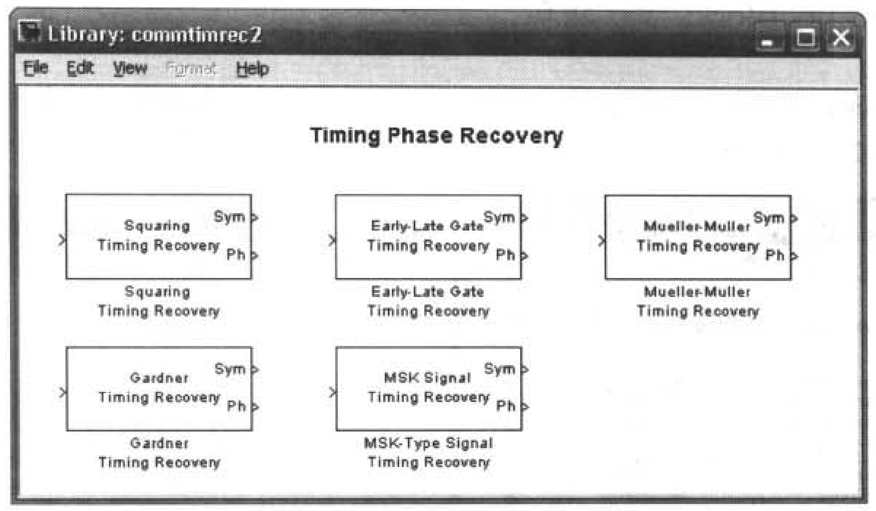


图 10-21 Timing Phase Recovery 模块组

10.2.7 Interleaving 子模块集

Interleaving 子模块集包括二个大类：Block 模块组和 Convolutional 模块组，如图 10-22 所示。

- Block Interleaving 模块组包括各种通用的交织、解交织模块，矩阵交织、解交织模块，代数交织、解交织模块，随机交织、解交织模块，如图 10-23 所示。
- Convolutional Interleaving 模块组包括卷积交织、解交织模块，复交织、解交织模块，螺旋交织、解交织模块，如图 10-24 所示。

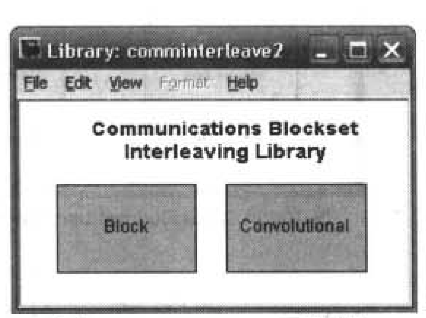


图 10-22 Interleaving 子模块集

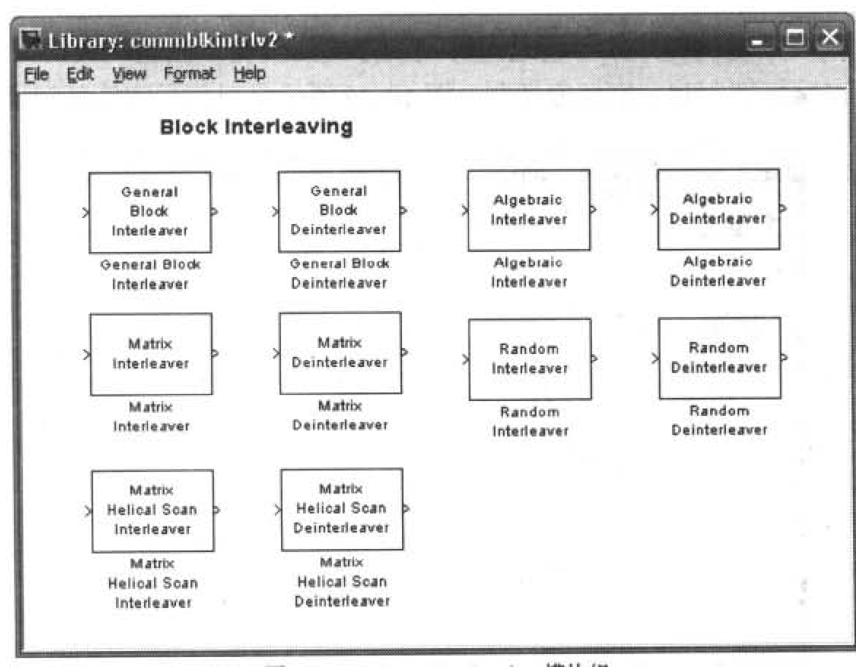
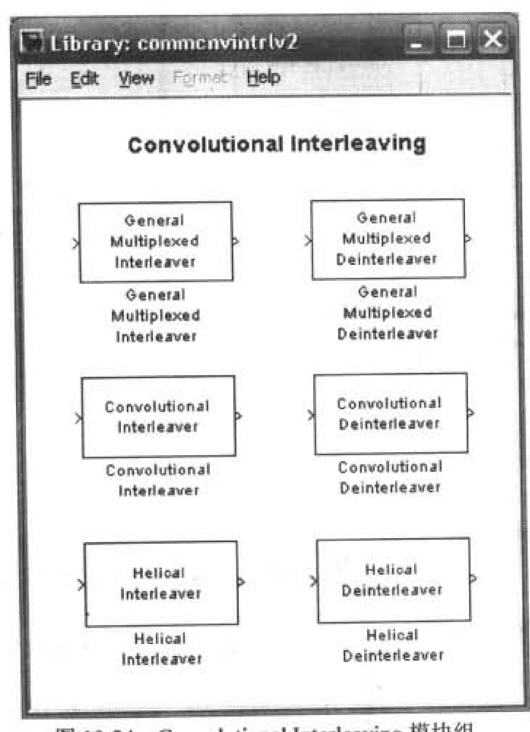


图 10-23 Block Interleaving 模块组

10.2.8 Utility Blocks 子模块集

Utility Blocks 子模块集包括各种常用的单元模块，如分贝转换模块、位与整型相互转换模块，如图 10-25 所示。



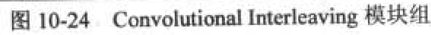


图 10-24 Convolutional Interleaving 模块组

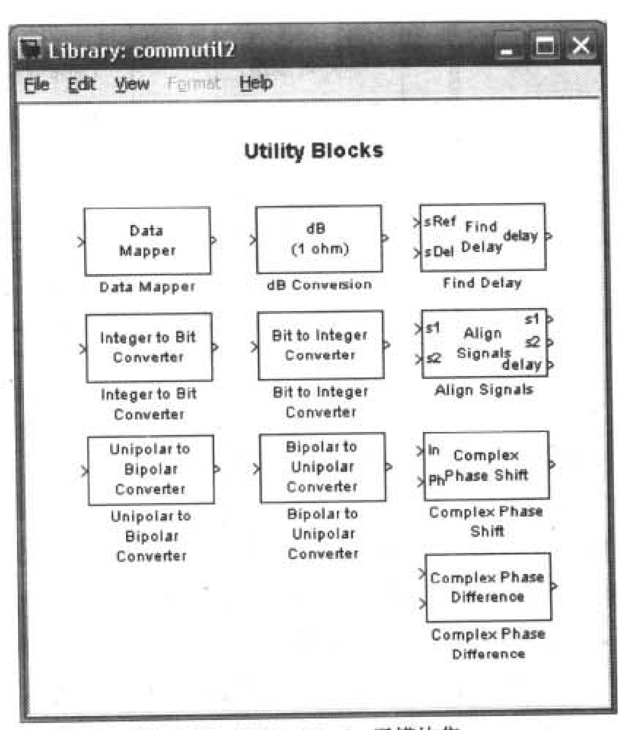


图 10-25 Utility Blocks 子模块集

10.3 通信系统仿真实例 1——数字幅度调制的抗噪声性能

本节我们将用一个实例程序来考察这两种调制方式的抗噪声性能。我们将创建两个仿真模型，分别用于实现脉幅调制和正交幅度调制。图 10-26 所示是对信号实施脉幅调制的仿真模型。

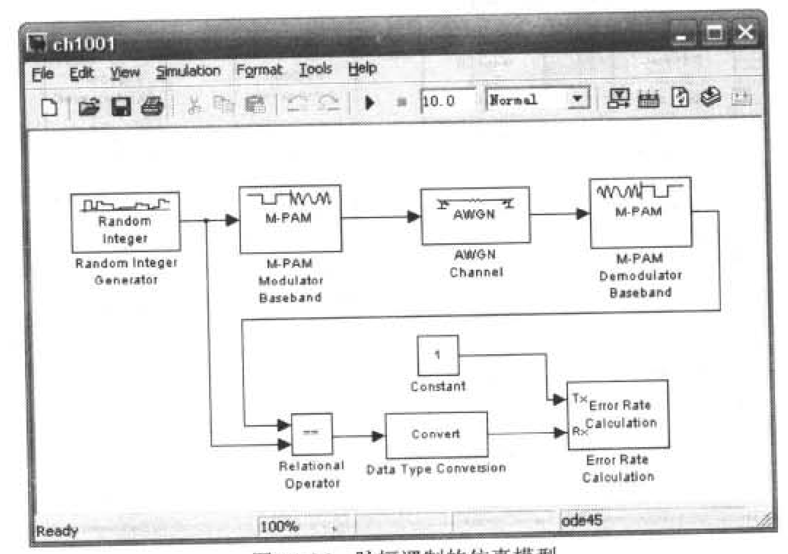


图 10-26 脉幅调制的仿真模型

在 PAM 调制的仿真模型中，Random Integer Generator（随机数产生器）产生一个八进制整数序列，这个整数序列通过 M-PAM Modulator（PAM 基带调制器模块）进行调制，得到基带调制信号。下面列出了随机整数产生器和 PAM 基带调制器模块的参数设置情况，其中 xSignalLevel、xInitialSeed 和 xSampleTime 分别表示整数序列的相数、随机整数产生器的初始化种子及其抽样间隔。

Random Integer Generator 参数设置如图 10-27 所示。

Parameters
M-ary number:
xSignalLevel
Initial seed:
xInitialSeed
Sample time:
xSampleTime
Frame-based outputs
Samples per frame:
1
Interpret vector parameters as 1-D

图 10-27 Random Integer Generator 参数设置

M-PAM Modulator Baseband 参数设置如图 10-28 所示。

Parameters
M-ary number:
xSignalLevel
Input type: Integer
Constellation ordering: Binary
Normalization method: Min. distance between symbols
Minimum distance:
2
Samples per symbol:
1

图 10-28 Modulator Baseband 参数设置

PAM 基带调制器模块产生的基带调制信号经过 AWGN Channel（加性白噪声信道）后叠加了一定强度的噪声，这个信号由 M-PAM Demodulator Baseband（PAM 基带解调器模块）进行解调，参数设置如下说明。

AWGN Channel 参数设置如图 10-29 所示。

Parameters
Initial seed:
37
Mode: Signal to noise ratio (SNR)
SNR (dB):
xSNR
Input signal power (watts):
1

图 10-29 AWGN Channel 参数设置

M-PAM Demodulator Baseband 参数设置如图 10-30 所示。

Parameters
M-ary number:
zSignalLevel
Output type: Integer
Constellation ordering: Binary
Normalization method: Min. distance between symbols
Minimum distance:
2
Samples per symbol:
1

图 10-30　M-PAM Demodulator Baseband 参数设置

最后，Error Rate Calculation（误码率统计模块）对原始信号和解调信号进行比较，统计得到 PAM 调制的误码率，并且把误码信息保存在 MATLAB 工作区变量 xErrorRate 中，Error Rate Calculation 参数设置如图 10-31 所示。

Parameters
Receive delay:
0
Computation delay:
0
Computation mode: Entire frame
Output data: Workspace
Variable name:
xErrorRate
Reset port
Stop simulation
Target number of errors:
100
Maximum number of symbols:
1e6

图 10-31　Error Rate Calculation 参数设置

到此为止，PAM 仿真模型的设计已经介绍完毕，QAM 调制的仿真模型与 PAM 调制模型非常相似，它只是把 PAM 基带调制器模块和解调器模块分别换成 Rectangular QAM Modulator Baseband（QAM 基带调制器模块）和 Rectangular QAM Demodulator Baseband（QAM 基带解调器模块），如图 10-32 所示。

在 QAM 调制的仿真模型中，信源、信道和信宿都与 PAM 仿真模型保持一致，这样便于在相同条件下比较两种调制方式的性能，下面分别列出了 QAM 基带调制器模块和 QAM 基带解调器模块的参数设置情况，如图 10-33 和图 10-34 所示。

为了比较两种调制方式在不同信噪比条件下的误码性能，需编写 M 文件 ch10_01.m,用于实现对仿真模型参数的初始化以及循环执行仿真模型。下面的程序段是 ch10_01.m 文件的内容。

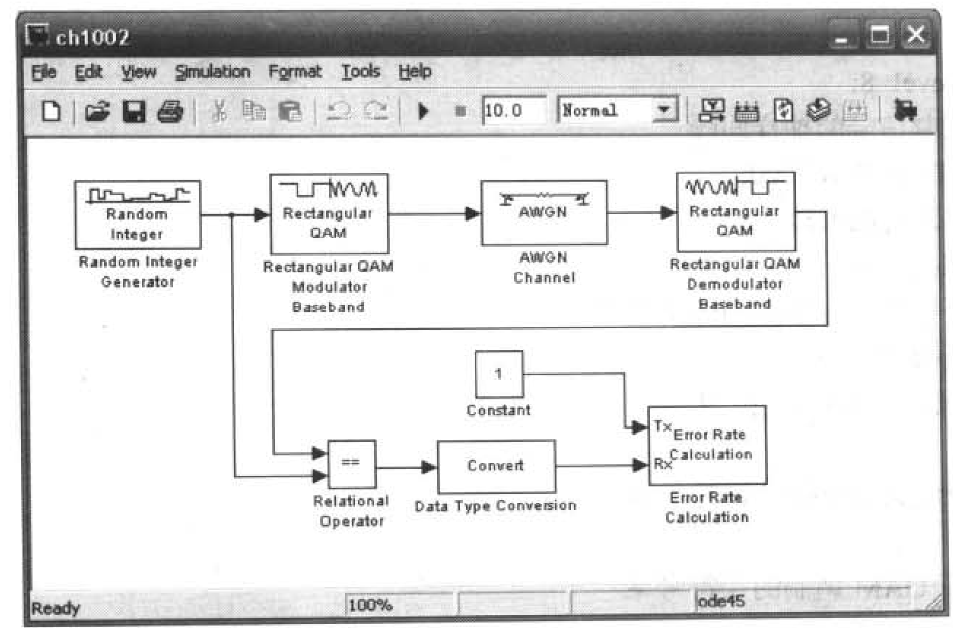


图 10-32　QAM 仿真模型

Parameters
M-ary number:
xSignalLevel
Input type: Integer
Constellation ordering: Binary
Normalization method: Min. distance between symbols
Minimum distance:
2
Phase offset (rad):
0
Samples per symbol:
1

图 10-33　QAM 基带调制器模块的参数设置

Parameters
M-ary number:
xSignalLevel
Output type: Integer
Constellation ordering: Binary
Normalization method: Min. distance between symbols
Minimum distance:
2
Phase offset (rad):
0
Samples per symbol:
1

图 10-34　QAM 基带解调器模块的参数设置

```
%设置调制信号的相数(调制信号是介于 0 和 xSignalLevel-1 之间的整数)
clear all;
```

```
clc;
xSignalLevel=8;
%设置调制信号的抽样间隔
xSampleTime=1/100000;
%设置仿真时间长度
xSimulationTime=10;
%设置随机数产生器的初始化种子
xInitialSeed=37;
%x 表示信噪比的取值范围
x=0:10;
%y1 表示 RAM 调制的误符号率
y1=x;
%y2 表示 QAM 调制的误符号率
y2=x;
for i=1:length(x);
  %信噪比依次取向量 x 的数值
  xSNR=x(i);
  %执行 RAM 仿真模型
  sim('ch1001');
  %从 xErrorRate 中获得调制信号的误符号率
  y1(i)=xErrorRate(1);
end
for i=1:length(x);
  %信噪比依次取向量 x 的数值
  xSNR=x(i);
  %执行 RAM 仿真模型
  sim('ch1002');
  %从 xErrorRate 中获得调制信号的误符号率
  y2(i)=xErrorRate(1);
end

%绘制信噪比与误符号率的关系曲线
%点表示 RAM 调制，三角表示 QAM 调制
semilogy(x,y1,'.',x,y2,'*');
```

仿真结束之后得到如图 10-35 所示的误码率曲线，其中圆点表示 PAM 调制误码率，星点表示 PAM 调制的误码率。从图 10-35 中可以看到，这两种调制方式的误码率是比较接近的，而 PAM 的抗噪声性能略优于 QAM。

另外，由于我们在仿真过程中把两种调制方式的抽样数（Samples per symbol 参数）设置为 1，因而仿真得到的误码率略高于理论计算数值，当增大 Samples per symbol 的数值时，PAM 和 QAM 的抗噪声性能随之增强，仿真得到的误码率将降低，并且逐渐趋向于理论计算值。

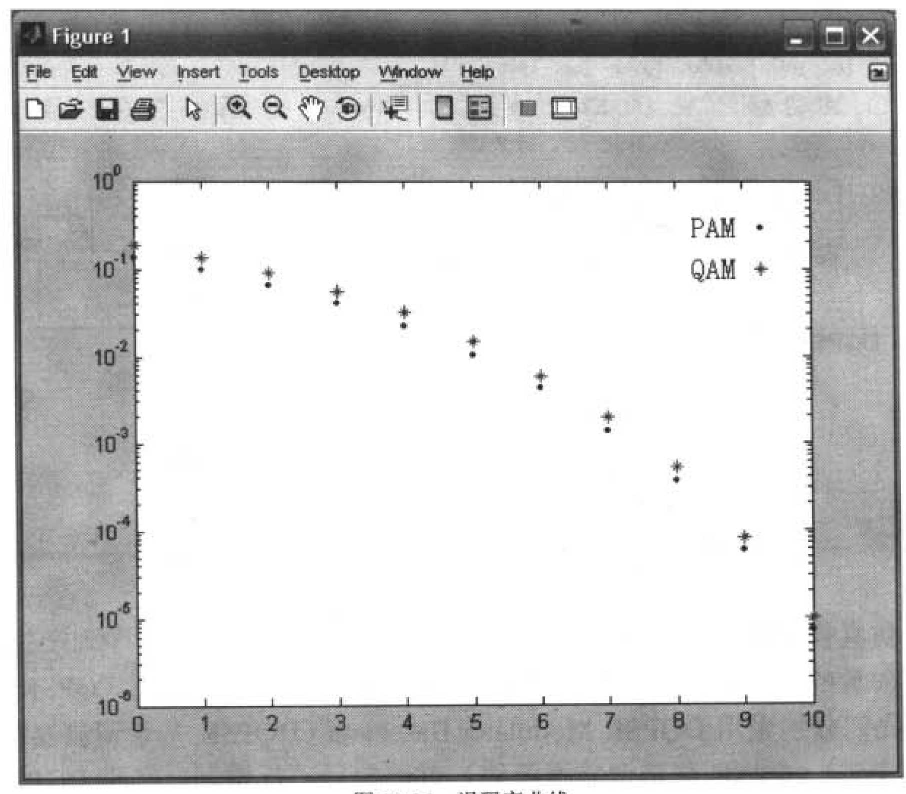


图 10-35　误码率曲线

10.4　通信系统仿真实例 2——QPSK 与 DQPSK 性能比较

本节我们设计一个 DQPSK 调制和解调系统的仿真模型和一个 QPSK 调制和解调系统的仿真模型，以考察二者的抗噪声性能，并对其误码率进行比较。该仿真模型的系统结构如图 10-36 和图 10-37 所示。

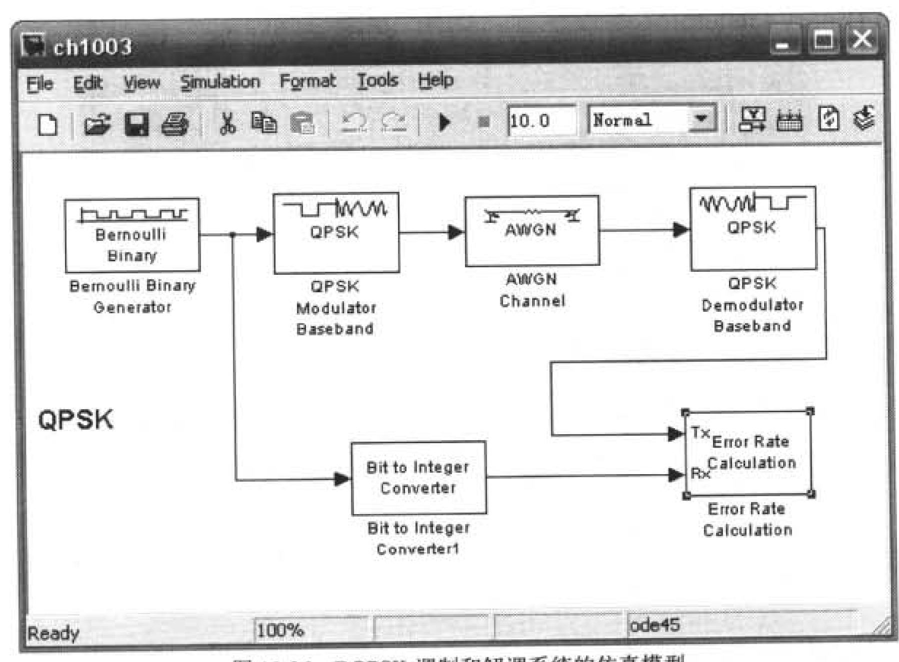


图 10-36　DQPSK 调制和解调系统的仿真模型

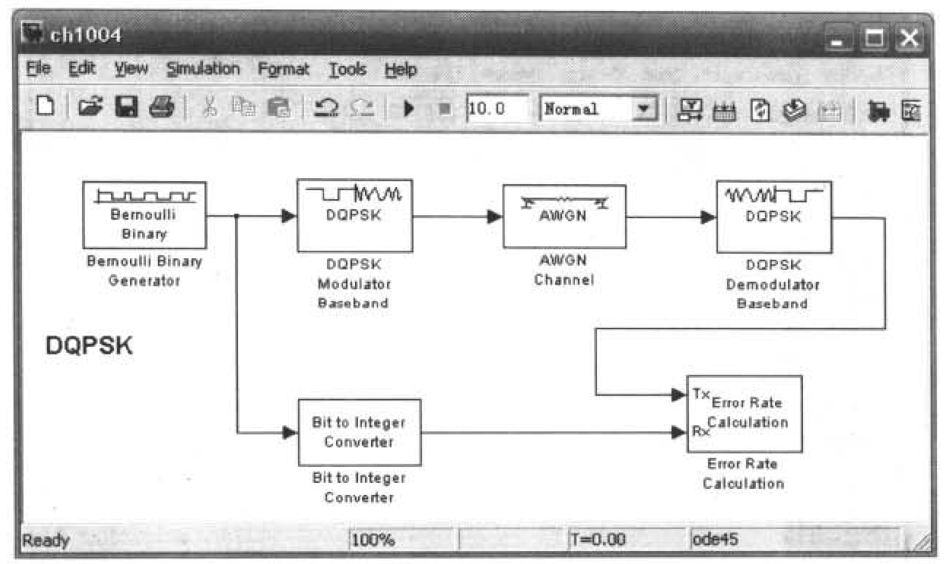


图 10-37 QPSK 调制和解调系统的仿真模型

在这两个仿真模型中，Bernoulli Binary Generator（贝努利二进制序列产生器）产生一个二进制向量，向量的长度等于 2，分别代表 QPSKModulator Baseband（QPSK 调制器）的两个输入信号。我们分别采用 DQPSK Modulator Baseband（DQPSK 基带调制器模块）和 QPSK Modulator Baseband（QPSK 基带调制器模块）对该信号进行调制，产生 DQPSK 基带调制信号和 QPSK 基带调制信号。基带调制信号经过 AWGN Channel（加性白噪声信道）后叠加了一定强度的噪声，然后分别让两个基带调制信号分别通过各自的解调器模块，对它们进行解调。二者解调器的参数设置保持一致，它们具有相同的相位偏移和抽样个数，并且都输出整数形式的解调信号。两个仿真模型中，贝努利二进制序列产生器产生的输出信号，经过 Bit to Integer Converter（数值转换模块）转换成整数后，各自的解调信号一起进入 Error Rate Calculation（误码率统计模块），二者误码率统计模块信号接收端的延时都等于 xReceiveDelay，同时把误码率的统计结果保存在工作区变量 xErrorRate 中。

下面给出各个模块的参数设置。

- Brnouli Binary Generator 参数设置，如图 10-38 所示。

Parameters

Probability of a zero:

0.5

Initial seed:

xInitialSeed

Sample time:

xSampleTime

☐ Frame-based outputs

Samples per frame:

1

☐ Interpret vector parameters as 1-D

图 10-38 Brnouli Binary Generator 参数设置

- QPSK Modulator Baseband 参数设置，如图 10-39 所示。

Parameters

Input type: Bit

Constellation ordering: Gray

Phase offset (rad):

xPhaseOffset

Samples per symbol:

xSamplesPerSymbol

图 10-39 QPSK Modulator Baseband 参数设置

- DQPSK Modulator Baseband 参数设置，如图 10-40 所示。

Parameters

Input type: Bit

Constellation ordering: Binary

Phase offset (rad):

xPhaseOffset

Samples per symbol:

xSamplesPerSymbol

图 10-40 DQPSK Modulator Baseband 参数设置

- AWGN Channel 参数设置，如图 10-41 所示。

Parameters

Initial seed:

67

Mode: Signal to noise ratio (SNR)

SNR (dB):

xSNR

Input signal power (watts):

1

图 10-41 AWGN Channel 参数设置

- QPSK Demodulator Baseband 参数设置，如图 10-42 所示。

Parameters

Output type: Integer

Constellation ordering: Binary

Phase offset (rad):

xPhaseOffset

Samples per symbol:

xSamplesPerSymbol

图 10-42 QPSK Demodulator Baseband 参数设置

- DQPSK Demodulator Baseband 参数设置，如图 10-43 所示。

Parameters

Output type: Integer

Constellation ordering: Binary

Phase offset (rad):

xPhaseOffset

Samples per symbol:

xSamplesPerSymbol

图 10-43　DQPSK Demodulator Baseband 参数设置

- Error Rate Calculation 的参数设置，如图 10-44 所示。

Parameters

Receive delay:

xReceiveDelay

Computation delay:

0

Computation mode: Entire frame

Output data: Workspace

Variable name:

xErrorRate

☐ Reset port

☐ Stop simulation

Target number of errors:

100

Maximum number of symbols:

1e6

图 10-44　Error Rate Calculation 的参数设置

为了得到 QPSK 调制与 DQPSK 调制的性能，编写 M 文件，对仿真模型中的各个变量进行赋值，然后依次改变信号的信噪比，循环执行仿真程序，两种调制都在相同条件下进行仿真，最后根据仿真的结果绘制曲线。M 文件代码如下：

```
%设置调制信号的抽样间隔
xSampleTime=1/100000;
%设置仿真时间长度
xSimulationTime=10;
%设置随机数产生器的初始化种子
xInitialSeed=[61 71];
%设置 QPSK 调制的初始相位
xPhaseOffset=pi/4;
%设置 QPSK 调制信号的抽样个数
xSamplesPerSymbol=1;
xReceiveDelay=0;
%x 表示信噪比的取值范围
x=0:10;
%y1 表示 RAM 调制的误符号率
y1=x;
```

```
%y2 表示 QAM 调制的误符号率
y2=x;
for i=1:length(x);
  %信噪比依次取向量 x 的数值
  xSNR=x(i);
  %执行 RAM 仿真模型
  sim('ch1005');
  %从 xErrorRate 中获得调制信号的误符号率
  y1(i)=xErrorRate(1);
  %执行 RAM 仿真模型
  sim('ch1003');
  %从 xErrorRate 中获得调制信号的误符号率
  y2(i)=xErrorRate(1);
end
%点表示 RAM 调制，三角表示 QAM 调制
semilogy(x,y1,'.',x,y2,'*');
```

在信噪比比较高的条件下，如果仿真的时间不够长，这时候仿真得到的误比特率通常为零。为此，我们把贝努利二进制序列产生器的抽样间隔 xSampleTime 设置为 1/100 000，同时把仿真时间 xSampleTime 设置为 10 秒，从而在一个仿真循环产生 10^6 个调制信号，以此提高仿真数据的精度。由此带来的另外一个问题是仿真需要较长的执行时间。仿真结果如图 10-45 所示。

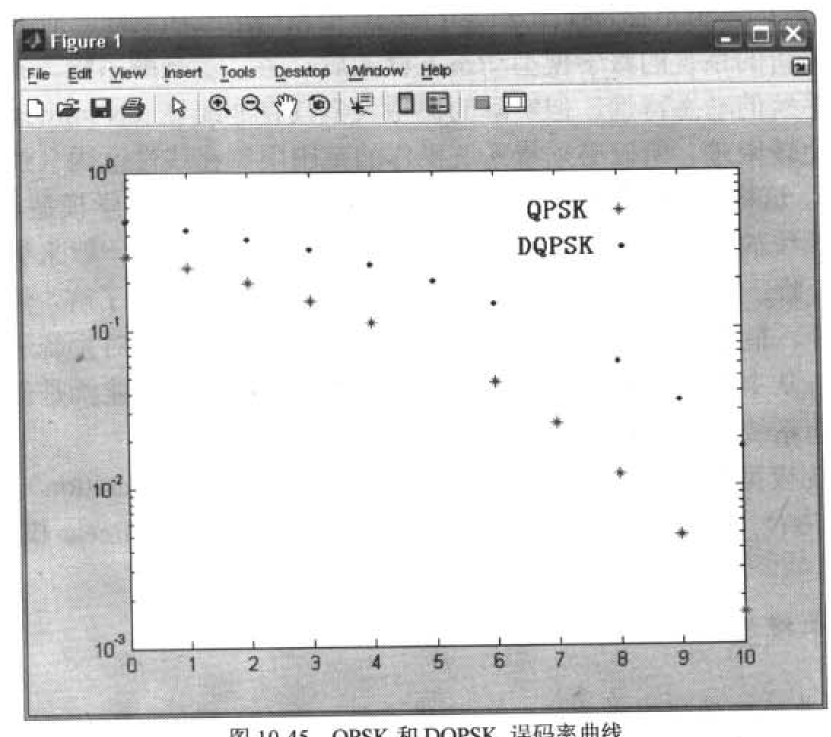


图 10-45　QPSK 和 DQPSK 误码率曲线

在图 10-45 所示中，实心圆点表示 QPSK 调制信号的误码率性能，星号则是 DQPSK 调制信号的误码率。从图中可以看到，在相同的条件下（相同的数据源、相同的信噪比以及相同的调制信号抽样数），QPSK 调制信号的性能优于 DQPSK。

第 11 章　Simulink6.0 在控制系统仿真中的应用

随着计算机的发明以及日益普及，越来越多的控制系统采用计算机进行控制。本章主要介绍 Simulink6.0 在控制系统仿真中的应用。

本章主要内容：

- 控制系统的模型
- 控制系统仿真实例 1——连续时间控制系统仿真
- 控制系统仿真实例 2——离散时间控制系统仿真

11.1　控制系统模型

分析与设计控制系统首要的工作是根据所要研究的系统的特性和相关定律，创建系统的数学模型。此系统的数学模型是一组方程式，可能是微分方程式，也可能是差分方程式，可能是线性的，也可能是非线性的，但不管怎样，这一组方程式要精确地或相当好地描述系统的动态特性。

得到描述要分析的系统的数学模型动态方程式后，第二步是解系统方程式以求得系统的响应，进而了解系统的动态特性。但实际的模型大多具有非线性（Nonlinear）的特性，因此求解系统方程式比较困难。所以必须将系统操作的范围限制在线性区内（线性系统理论已经发展得相当完备），这样控制工程师不仅有能力准确地描述系统的数学模型，更重要的是能正确地估计与假设系统的线性区域，以便以线性模型来分析此系统。一般来说，可以先创建一个简化的模型（低阶、线性的），求得对系统的动态行为的一个初步了解，然后再创建更复杂的数学模型（高阶、非线性的），当然要更能代表系统的特性，用来对系统进行比较精确的分析。因此对于有志从事控制工程的人而言，首先要求的两个基本技能就是创建动态系统的数学模型及分析动态系统响应的能力。

创建线性系统模型最常用的两种方法是传递函数（Transfer Function）法与动态方程式（Dynamical Equation）法，这两种方法在 Simulink 中都有提供（在 linear 模块库内）。

11.1.1　数学模型

1. 经典控制学的数学模型表示法——传递函数

定义：假设一个动态系统，根据其物理特性可推导出代表动态特性的线性非时变常微分方程：

$$y^{(n)} + a_1 y^{(n-1)} + \cdots + a_n y = b_0 u^{(m)} + b_1 u^{(m-1)} + \cdots + b_m u \qquad (11.1)$$

其中 y 为系统输出，u 为系统输入，$a_1,\cdots,a_n$，$b_0,\cdots,b_m \in R$ 为系统参数，令系统的初始条件都为零，即：

$$y^{(n-1)}(0)=y^{(n-2)}(0)=\cdots=y(0)=u^{(m-1)}(0)=\cdots=u(0)=0$$

对式 11.1 取 Laplace 变换，并定义 $Y(s)=L\{y(t)\}$，$U(s)=L\{u(t)\}$，则可得：

$$s^nY(s)+a_1s^{(n-1)}Y(s)+\cdots+a_nY(s)=b_0s^mU(s)+\cdots+b_mu(S) \tag{11.2}$$

将式 11.1 重新整理可得：

$$G(s)=\frac{Y(s)}{U(s)}=\frac{b_0s^m+\cdots+b_m}{s^n+a_1s^{n-1}+\cdots+a_n} \tag{11.3}$$

$G(s)$定义为此动态系统输出与输入间的传递函数。

- 由定义可知传递函数是系统输出与输入之间 Laplace 变换的比值，传递函数本身只与微分方程的系数 $a_1,\cdots,a_n$，$b_0,\cdots,b_m$ 有关，即与系统的动态（物理）特性有关，同系统的输出、输入没有关系。
- 能以传递函数表示的动态系统必须是线性非时变且初始值为零的系统。
- 传递函数也可表示为零极点（Pole-Zero）形式：

$$G(s)=\frac{K(s+z_1)\cdots(s+z_m)}{(s+p_1)(s+p_2)\cdots(s+p_n)}$$

其中，K 称为系统增益常数（Gain Constant）。

$s=-z_1,\cdots,-z_m$ 称为系统零点（Zero）。

$s=-p_1,\cdots,-p_n$ 称为系统极点（Pole）。

2. 现代控制学的模型表示法——动态方程式

上面所述的传递函数表示法适用于单输入单输出线性非时变系统的分析与设计。但近代控制工程趋向于多元化，且系统精确度要求不断提高，也更加复杂，使得控制工程师必须面对多输入多输出和时变系统。要分析这类系统，必须降低数学表达式的复杂性，又由于计算机科技的急速发展，可做大量的数据处理与计算，而动态方程式表示法易于表达多输入多输出和时变观念，且适于计算机的计算，因此它在这方面扮演了重要的角色。

对于一个具有 p 个输入 $u_1,u_2,\cdots,u_p$，q 个输出 $y_1,y_2,\cdots,y_q$，及 n 个状态变量 $x_1,x_2,\cdots,x_n$ 的线性非时变系统，将每一个状态变量的微分表示为所有状态变量与输入的线性组合，称为状态方程式（State Equation）。而且将每一个输出表示为所有状态变量与输入的线性组合，称为输出方程式（Output Equation）。状态方程式和输出方程式合称为动态方程式。

现在定义输入、输出与状态变量向量为列矩阵（Column Matrixes）形式：

$$U(t)=\begin{bmatrix}u_1(t)\\u_2(t)\\\vdots\\u_p(t)\end{bmatrix}(p\times1)$$

$$Y(t)=\begin{bmatrix}y_1(t)\\y_2(t)\\\vdots\\y_q(t)\end{bmatrix}(q\times1)$$

$$X(t)=\begin{bmatrix}x_1(t)\\x_2(t)\\\vdots\\x_n(t)\end{bmatrix}(n\times1)$$

动态方程式可写成：

$$\dot{X}(t)=AX(t)+BU(t)$$
$$Y(t)=CX(t)+DU(t)$$

其中 A 为 $n\times n$ 阶系统矩阵（System Matrix）：

$$A=\begin{pmatrix}a_{11}&\cdots&a_{1n}\\\vdots&\ddots&\vdots\\a_{n1}&\cdots&a_{nn}\end{pmatrix}$$

B 为 $n\times p$ 阶输入矩阵（Input Matrix）

$$B=\begin{pmatrix}b_{11}&\cdots&b_{1p}\\\vdots&\ddots&\vdots\\b_{n1}&\cdots&b_{np}\end{pmatrix}$$

C 为 $q\times n$ 阶输出矩阵（Output Matrix）

$$C=\begin{pmatrix}c_{11}&\cdots&c_{1n}\\\vdots&\ddots&\vdots\\c_{q1}&\cdots&c_{qn}\end{pmatrix}$$

D 为 $q\times p$ 阶直接传输矩阵（Direct Transmission Matrix）：

$$D=\begin{pmatrix}d_{11}&\cdots&d_{1p}\\\vdots&\ddots&\vdots\\d_{q1}&\cdots&d_{qp}\end{pmatrix}$$

动态方程式可用于单输入单输出系统，也可用于多输入多输出系统。若系统是非时变系统，则 A、B、C、D 均为常数（Constant）矩阵。若系统是时变系统，则 $A(t)$、$B(t)$、$C(t)$、$D(t)$均为时变（Time Varying）矩阵。

11.1.2 数学模型转换（删除）性能指标

MATLAB 提供一些函数作为数学模型间的转换，如下所述：

- ss2tf：动态方程式表达式到传递函数表达式间的转换。
- ss2zp：动态方程式表达式到零极点表达式间的转换。
- tf2ss：传递函数表达式到动态方程式表达式间的转换。
- tf2zp：传递函数表达式到零极点表达式间的转换。
- zp2ss：零极点表达式到动态方程式表达式间的转换。
- zp2tf：零极点表达式到传递函数表达式间的转换。

1. ss2tf 函数

ss2tf 函数能将连续时间动态方程式：

$$\dot{X}(t)=AX(t)+BU(t)$$
$$Y(t)=CX(t)+DU(t)$$

转换为传递函数格式：

$$G(s)=\frac{Y(s)}{U(s)}=\frac{b_0 s^m+\cdots+b_m}{s^n+a_1 s^{n-1}+\cdots+a_n}$$

命令格式：

[nums，dens]=ss2tf(A,B,C,D,iu)

A、*B*、*C*、*D* 分别代表动态方程式中的系统矩阵、输入矩阵、输出矩阵和直接传输矩阵。因为动态方程式能表示成多输入系统，iu 值代表求第 i_th 个输入的传递函数表达式。nums、dens 分别代表传递函数分子、分母多项式的系数（以 s 的降幂阶排列）。

举例：有一个系统以下列动态方程描述：

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\end{bmatrix}=\begin{pmatrix}0 & 1 & 0\\ 0 & 0 & 1\\ -1 & -2 & -3\end{pmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix}+\begin{bmatrix}5\\ 0\\ 0\end{bmatrix}u$$

$$y=\begin{bmatrix}1 & 0 & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix}$$

试求传递函数 $G(s)=y(s)/U(s)$

在 MATLAB 窗口中从键盘输入：

```
A=[0 1 0;0 0 1;-1 -2 -3];
B=[5;0;0]
C=[1 0 0];
D=[0];
[nums,dens]=ss2tf(A,B,C,D,1)
```

执行结果如下：

```
nums=
        0      5.0000      15.0000      10.0000
dens=
     1.0000    3.0000      2.0000       1.0000
```

因此传递函数为：

$$G(s)=\frac{5s^2+15s+10}{s^3+3s^2+2s+1}$$

2. ss2zp 函数

ss2zp 函数能将连续时间动态方程式：

$$\dot{X}(t)=AX(t)+BU(t)$$
$$Y(t)-CX(t)+DU(t)$$

转换为零极点表达式：

$$G(s)=\frac{K(s+z_1)\cdots(s+z_m)}{(s+p_1)(s+p_2)\cdots(s+p_n)}$$

命令格式：

[z,p,k]=ss2zp(A,B,C,D,iu)

A、B、C、D 分别代表动态方程式中的系数矩阵、输入矩阵、输出矩阵和直接传输矩阵。因为动态方程式能表示成多输入系统，iu 值代表求第 i_th 个输入的传递函数表达式。z、p、k 分别代表极点、零点、增益值。

举例：

有一个系统以下列动态方程式描述：

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix}=\begin{pmatrix}0 & 1\\ -3 & -4\end{pmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}+\begin{bmatrix}0\\ 1\end{bmatrix}u$$

$$y=[10 \quad 0]\begin{bmatrix}x_1\\ x_2\end{bmatrix}+[0]u$$

试求以零极点表示的传递函数 $G(s)$。

在 MATLAB 命令窗口中输入：

```
A=[0 1;-3 -4]
B=[0;1]
C=[10 0];
D=[0];
[z,p,k]=ss2zp(A,B,C,D,1)
```

执行结果如下：

```
z=
    Empty matrix:0-by-1
p=
      -1
      -3
k=
      10
```

因此以零极点表示的传递函数为：

$$G(s)=\frac{10}{(s+1)(s+3)}$$

3. tf2ss 函数

tf2ss 函数能将连续时间传递函数：

$$G(s)=\frac{Y(s)}{U(s)}=\frac{b_0s^m+\cdots+b_m}{s^n+a_1s^{n-1}+\cdots+a_n}$$

转换为动态方程式表达式：

$$\dot{X}(t) = AX(t) + BU(t)$$
$$Y(t) = CX(t) + DU(t)$$

命令格式：

[A,B,C,D]=tf2ss(nums,dens)

A、B、C、D 分别代表动态方程式中的系数矩阵、输入矩阵、输出矩阵和直接传输矩阵。nums、dens 分别代表传递函数分子、分母多项式系数（以 s 的幂次降阶排列）。

举例：

试求下列传递函数的动态方程式表达式。

$$G(s) = \frac{s^2 + 7s + 2}{s^3 + 9s^2 + 26s + 24}$$

在 MATLAB 命令窗口输入：

```
nums=[1 7 2];
dens=[1 9 26 24];
[A，B，C，D]=tf2ss(nums,dens)
```

执行结果如下：

```
A=
    -9    -26    -24
     1      0      0
     0      1      0
B=
     1
     0
     0
C=
     1    7    2
D=
     0
```

4. tf2zp 函数

tf2zp 函数能将连续时间系统传递函数：

$$G(s) = \frac{Y(s)}{U(s)} = \frac{b_0 s^m + \cdots + b_m}{s^n + a_1 s^{n-1} + \cdots + a_n}$$

转换为零极点表达式：

$$G(s) = \frac{K(s + z_1)\cdots(s + z_m)}{(s + p_1)(s + p_2)\cdots(s + p_n)}$$

命令格式：

[z,p,k]=tf2zp(nums,dens)

z、p、k 分别代表极点、零点和增益值。nums、dens 分别代表传递函数分子和分母多项式的系数（以 s 的幂次降阶排列）。

举例：

试求下列传递函数的零极点表达式：

$$G(s)=\frac{s^2+7s+2}{s^3+9s^2+26s+24}$$

在 MATLAB 命令窗口输入：

```
nums=[1 7 2];
dens=[1 9 26 24];
[z,p,k]=tf2zp(nums,dens)
```

执行结果如下：

```
z=
   -6.7016
   -0.2984
p=
   -4.0000
   -3.0000
   -2.0000
k=
      1
```

5. zp2ss 函数

zp2ss 函数能将以极零点表示的传递函数：

$$G(s)=\frac{K(s+z_1)\cdots(s+z_m)}{(s+p_1)(s+p_2)\cdots(s+p_n)}$$

转换为动态方程表达式：

$$\dot{X}(t)=AX(t)+BU(t)$$
$$Y(t)=CX(t)+DU(t)$$

命令格式：

[A,B,C,D]=zp2ss(z,p,k)

A、B、C、D 分别代表动态方程式中的系数矩阵、输入矩阵、输出矩阵和直接传输矩阵。z、p、k 分别代表极点、零点和增益值。

举例：

试求下列以零极点表示的传递函数的动态方程表达式：

$$G(s)=\frac{5}{(s+2)(s+3)}$$

在 MATLAB 命令窗口中输入：

```
z=[];
p=[-2 -3];
k=5;
[A,B,C,D]=zp2ss(z,p,k)
```

执行结果如下：

```
A=
    -5.0000    -2.4495
     2.4495          0
B=
     1
     0
C=
          0    2.0412
D=
     0
```

6. zp2tf 函数

zp2tf 函数能将以零极点表示的传递函数：

$$G(s)=\frac{K(s+z_1)\cdots(s+z_m)}{(s+p_1)(s+p_2)\cdots(s+p_n)}$$

转换为动态方程表达式：

$$G(s)=\frac{Y(s)}{U(s)}=\frac{b_0s^m+\cdots+b_m}{s^n+a_1s^{n-1}+\cdots+a_n}$$

命令格式：

[nums,dens]=zp2tf(z,p,k)

nums、dens 分别代表传递函数分子和分母多项式的系数（以 s 的幂次降阶排列）。z、p、k 分别代表极点、零点和增益值。

举例：

将下列零极点表示的传递函数转换为传递函数表达式：

$$G(s)=\frac{s(s+5)(s+6)}{(s+1)(s+2)(s+3+4j)(s+3-4j)}$$

在 MATLAB 命令窗口输入：

```
z=[0;-5;-6];
i=sqrt(-1);
p=[-1;-2;-3+4i;-3-4i];
k=1;
[nums,dens]=zp2tf(z,p,k)
```

执行结果如下：

```
nums=
     0     1     11     30     0
dens=
     1     9     45     87     50
```

一次传递函数为：

$$G(s)=\frac{s^3+11s^2+30s}{s^4+9s^3+45s^2+87s+50}$$

11.2　控制系统仿真实例1——连续时间控制系统仿真

直流电机是工业上应用最广泛的电机之一，直流电机具有良好的调整速度特性，较大的起动转矩及功率大、响应快等优点。在伺服系统中应用的直流电机称为直流伺服电机，小功率的直流伺服电机往往应用在磁盘驱动器的驱动及打印机等与计算机相关的设备中，大功率的直流伺服电机则往往应用在工业机器人系统和OCNC铣床等大型工具上。

1．直流伺服电机的电枢控制

直流伺服电机一般包括3个组成部分：

- 磁极

电机的定子部分，有磁铁N－S极组成，可以是永久的磁铁（此类称为永磁式直流伺服电机），也可以是由绕在磁极上的激磁线圈构成。

- 电枢

电机的转子部分，为表面上绕有线圈的圆柱形铁芯，线圈与换向片焊接在一起。

- 电刷

电机定子的一部分，当电枢转动时，电刷交替地与换向片接触在一起。

本节所介绍的直流伺服电机，其中激磁电流保持常数，而由电枢电流进行控制。这种利用电枢电流对直流伺服电机的输出速度的控制称为直流伺服电机的电枢控制。如图11-1所示。

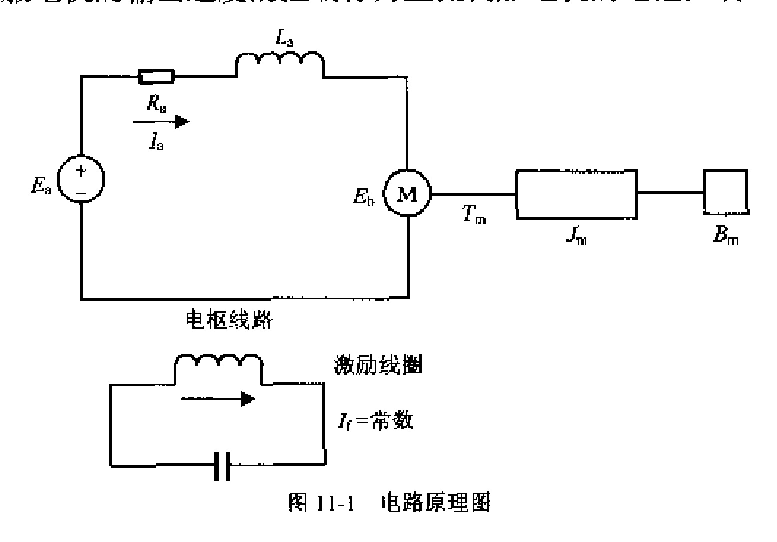


图11-1　电路原理图

其中，

- E_a定义为电枢电压（伏特）
- I_a定义为电枢电流（安培）
- R_a定义为电枢电阻（欧姆）
- L_a定义为电枢电感（亨利）
- E_b定义为反电动势（伏特）
- I_f定义为激磁电流（安培）

- θ_m 定义为转轴角位移（弧度）
- T_m 定义为电机产生的转矩（牛顿. 米）
- B_m 定义为电机和反射到电机轴上的负载的等效粘滞系数（牛顿. 米/弧度. 秒$^{-1}$）
- J_m 定义为电机和反射到电机轴上的负载的等效转动惯量（千克. 米^2）

电机所产生的转矩 T_m，正比于电枢电流 I_a 与气隙磁通 Φ 的乘积，即：

$$T_m = K_1^n I_a \Phi \tag{11.4}$$

而气隙磁通 Φ 又正比于激磁电流 I_f 故式（11.4）改写为：

$$T_m = K_1^n I_a K_f I_f = K I_a \tag{11.5}$$

电流 I_f 为常数，$K_1 K_f I_f$ 合并为一个常数 K，称为电机力矩常数。电枢电流 I_a 的正负值即代表电机的正反转。

当电枢转动时，在电枢中感应出与电机转轴角速度成正比的电压，称为反电动势，即：

$$E_b = K_b \omega_m = K_b \frac{d\theta_m}{dt} \tag{11.6}$$

其中，K_b 称为反电动势常数。

电机的速度是由电枢电压 E_a 控制，应用基尔霍夫电压定律导出电枢电流 I_a 的微分方程式为：

$$L_a \frac{dI_a}{dt} + R_a I_a + E_b = E_a \tag{11.7}$$

电枢电流 I_a 产生的力矩，用来克服系统所含负载的惯性和摩擦，可得

$$J_m \frac{d^2\theta_m}{dt^2} + B_m \frac{d\theta_m}{dt} = T = K I_a \tag{11.8}$$

根据式（11.6）、式（11.7）、式（11.8），在 Simulink 中所建造的模型如图 11-2 所示。

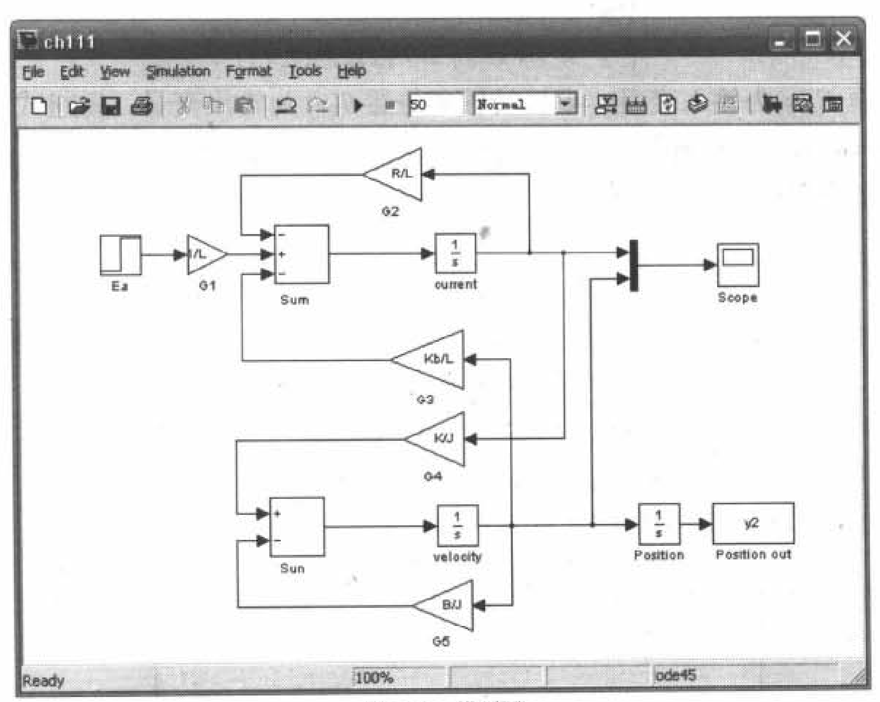


图 11-2 模型图

令 R_a=1、L_a=0.2、K_b=1、B_m=0.1、J_m=5、K=0.5，在时间 1 秒加入 2 伏特驱动电压，由 Scope 模块所观察的输出波形如图 11-3 所示。实线轨迹为电枢电流波形，虚线轨迹为电机

转速波形。

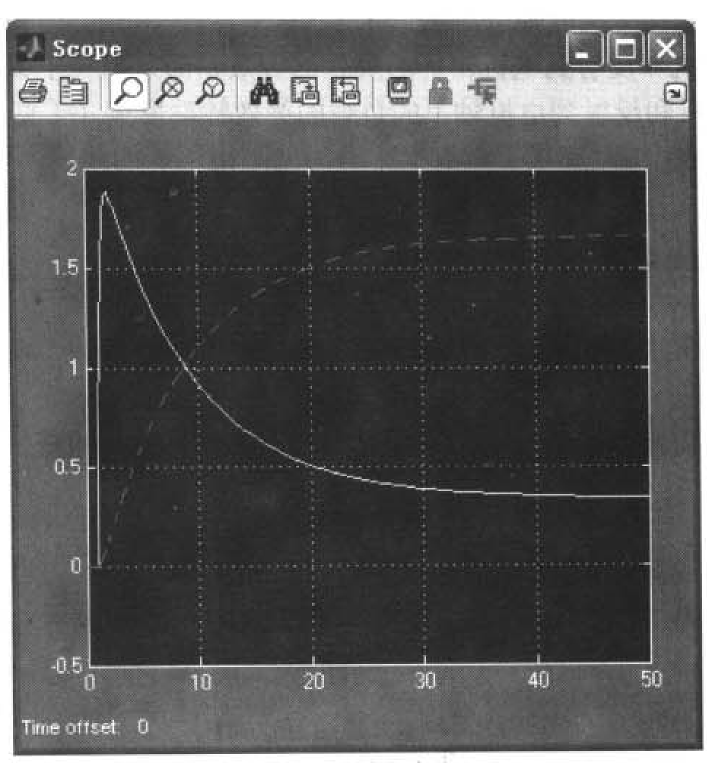


图 11-3 输出波形

假设所有变量的初始值都为零，对式（11.6）、式（11.7）、式（11.8）分别取拉普拉斯变换，可得下列方程式：

$$\begin{aligned} &E_b(s) = K_b s\theta(s) \\ &(L_a s + R_a)I_a(s) + E_b(s) = E_a(s) \\ &(Js^2 + Bs)\theta(s) = T(s) = KI_a(s) \end{aligned} \tag{11.9}$$

设电枢电压 $E_a(s)$为输入变量，电机转轴角位置 $\theta(s)$ 为输出变量。重组式（11-9）可得电机系统的框图（如图 11-4 所示），反电动势可以看作是一个与电机速度成比例的反馈信号，它增加了系统的有效阻尼（Damping）。上述直流伺服电机的传递函数为：

$$\frac{\theta(s)}{E_a(s)} = \frac{K}{s[JL_a s^2 + L_a B + R_a J)s + R_a B + KK_b]} \tag{11.10}$$

如果电枢电路中的电感 L_a 小到可以忽略不计，则式（11.10）的传递函数可以简化为：

$$\frac{\theta(s)}{E_a(s)} = \frac{K/(R_a B + KK_b)}{[R_a J/(R_a B + KK_b)]s^2 + s} = \frac{K_m}{s(T_m s + 1)} \tag{11.11}$$

K_m 定义为电机的增益常数，T_m 定义为电机的时间常数。

依上述表示的 DC 电机框图如图 11-4 所示。

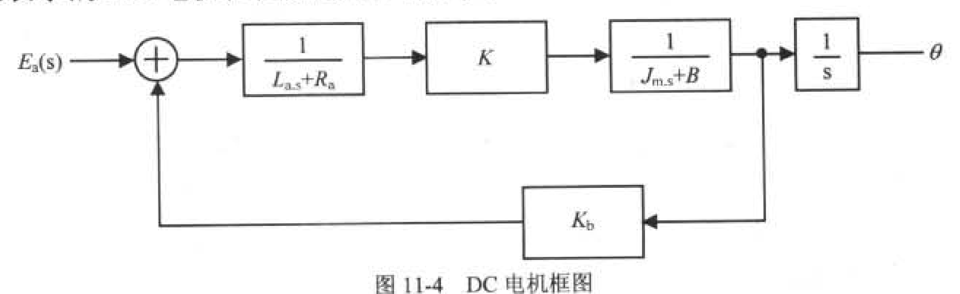


图 11-4 DC 电机框图

在 Simulink 中应用 Transfer Fcn 模块所建构的模型如图 11-5 所示。

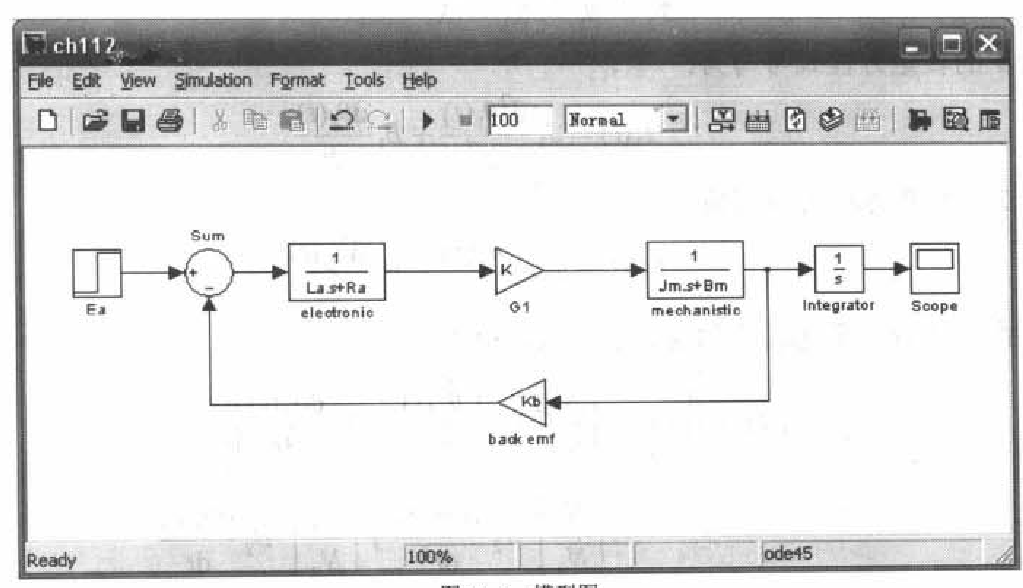


图 11-5 模型图

2. 包含齿轮传动的系统

齿轮依靠转矩的改变，能使能量由一个系统传到另一个系统。如图 11-6 所示两个齿轮耦合在一起，考虑实际状况，耦合的齿轮都存在摩擦和惯量。以下定义如图 11-6 所示齿轮系统的参数：

- T：外加转矩。
- T_1,T_2：齿轮间传送的转矩。
- θ_1,θ_2：角位移。
- R_1,R_2：齿轮的半径。
- J_1,J_2：齿轮的惯量。
- B_1,B_2：粘滞摩擦系数。
- N_1,N_2：齿数。
- ω_1,ω_2：角速度

图 11-6 齿轮传动的系统

推导过程如下：

齿轮的齿数与齿轮半径成正比，即：

$$\frac{N_1}{R_1}=\frac{N_2}{R_2} \tag{11.12}$$

各齿轮沿其接触面移动的距离均相等，即：

$$\theta_1 R_1=\theta_2 R_2 \tag{11.13}$$

若无摩擦存在，则无能量损耗。由一个齿轮传送的功，等于另一各齿轮收到的功，即：

$$T_1\theta_1=T_2\theta_2 \tag{11.14}$$

齿轮的角位移与角速度成正比，即：

$$\frac{\theta_1}{\omega_1}=\frac{\theta_2}{\omega_2} \tag{11.15}$$

综合式（11.12）、式（11.13）、式（11.14）、式（11.15）可得：

$$\frac{T_1}{T_2}=\frac{\theta_2}{\theta_1}=\frac{\omega_2}{\omega_1}=\frac{R_1}{R_2}=\frac{N_1}{N_2} \tag{11.16}$$

齿轮 1 的转矩方程式可写为：

$$T_1(t)=J_1\frac{\mathrm{d}^2\theta_1(t)}{\mathrm{d}t^2}+B_1\frac{\mathrm{d}\theta_1(t)}{\mathrm{d}t}$$

齿轮 2 的转矩方程可以写为：

$$T_2(t)=J_2\frac{\mathrm{d}^2\theta_2(t)}{\mathrm{d}t^2}+B_2\frac{\mathrm{d}\theta_2(t)}{\mathrm{d}t}$$

应用式（11.16）所述转矩与齿数的关系，可得：

$$\begin{aligned}T_1(t)=\frac{N_1}{N_2}T_2&=\left[\frac{N_1}{N_2}\left(J_2\frac{\mathrm{d}^2\theta_2(t)}{\mathrm{d}t^2}+B_2\frac{\mathrm{d}\theta_2(t)}{\mathrm{d}t}\right)\right]\\&=\left(\frac{N_1}{N_2}\right)^2J_2\frac{\mathrm{d}^2\theta_1(t)}{\mathrm{d}t^2}+\left(\frac{N_1}{N_2}\right)^2B_2\frac{\mathrm{d}\theta_1(t)}{\mathrm{d}t}\end{aligned}$$

由 $T(t)=T_1(t)+T_2(t)$得

$$T(t)=\left(J_2+\left(\frac{N_1}{N_2}\right)^2J_2\right)\frac{\mathrm{d}^2\theta_1(t)}{\mathrm{d}t^2}+\left(B_2+\left(\frac{N_1}{N_2}\right)^2B_2\right)\frac{\mathrm{d}\theta_1(t)}{\mathrm{d}t}$$

由此可知，由齿轮 2 反射到齿轮 1，整理可得下列各量：

- 惯量：$\left(\frac{N_1}{N_2}\right)^2J_2$
- 粘滞摩擦系数：$\left(\frac{N_1}{N_2}\right)^2B_2$
- 转矩：$\frac{N_1}{N_2}T_2$
- 角速度：$\frac{N_2}{N_1}\omega_2$
- 角位移：$\frac{N_2}{N_1}\theta_2$

11.3 控制系统仿真实例 2——离散时间控制系统仿真

本节介绍自动引导车系统平面运动模型，通过运动误差方程导出自动引导车的状态空间模式，以状态空间表示式来设计路径追踪调节器。以 MATLAB Simulink 仿真，结果证明这个调节器的使用可以消除轨迹的误差，使自动引导车具有准确跟随路径的能力。

自动引导车在平面上，沿一设计好的需求轨迹运动，因内部因素如电机的响应速度无法匹配或负载不平衡及外部因素，如轮子打滑等，使得自动引导车偏离已设置好的轨迹，这样就会有误差产生，定义 3 种运动误差。

- dx 侧向追踪误差

- dy　前向追踪误差
- $d\theta$　指向追踪误差

如图 11-7 所示，由机器人两驱动轮的速度及需求线速度及角速度可推导出以下的误差方程式：

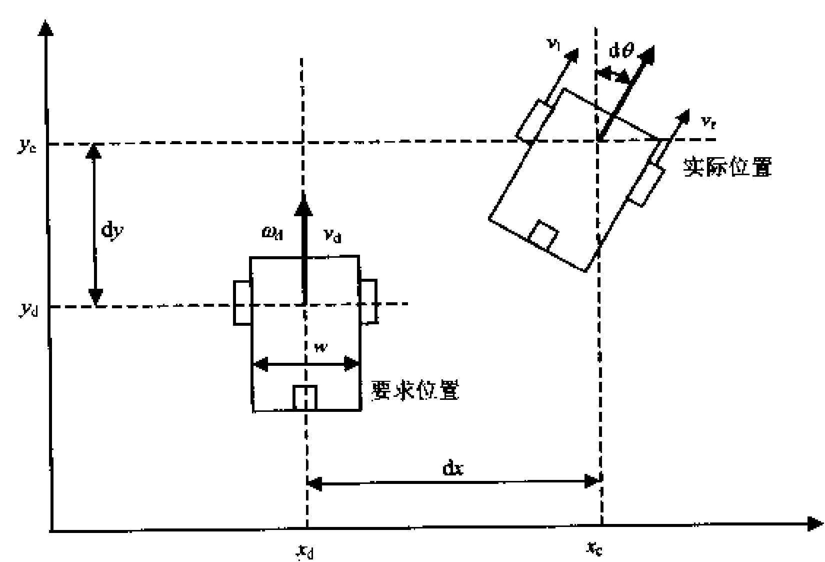


图 11-7　自动引导车系统平面运动模型

$$d\dot{\theta}=\frac{v_l-v_r}{w}+\omega_d \tag{11.17}$$

$$d\dot{x}=\left(\frac{v_l+v_r}{2}\right)\sin(d\theta) \tag{11.18}$$

$$d\dot{y}=\left(\frac{v_l+v_r}{2}\right)\cos(d\theta)-v_d \tag{11.19}$$

其中，

- w：两驱动轮间距离
- v_d：要求的线速度
- ω_d：要求的角速度

当沿轨迹行进中，指向误差 $d\theta$ 非常小，即 $|d\theta|<<1$ 所以 $\sin(d\theta)$ 可近似等于 $d\theta$，$\cos(d\theta)$ 可近似等于 1。故上面 3 个式子可线性化为：

$$d\dot{\theta}=\frac{v_l-v_r}{w}+\omega_d \tag{11.20}$$

$$d\dot{x}=\left(\frac{v_l+v_r}{2}\right)(d\theta) \tag{11.21}$$

$$d\dot{y}=\left(\frac{v_l+v_2}{2}\right)-v_d \tag{11.22}$$

式（11.20）、式（11.21）、式（11.22）用状态方程式，可写为：

$$\begin{bmatrix} d\dot{\theta} \\ d\dot{x} \\ d\dot{y} \end{bmatrix}=\begin{bmatrix} 0 & 0 & 0 \\ v_c & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} d\theta \\ dx \\ dy \end{bmatrix}+\begin{bmatrix} \frac{1}{w} & \frac{-1}{w} \\ 0 & 0 \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}\begin{bmatrix} v_l \\ v_r \end{bmatrix}+\begin{bmatrix} 0 & 1 \\ 0 & 0 \\ -1 & 0 \end{bmatrix}\begin{bmatrix} v_d \\ \omega_d \end{bmatrix} \tag{11.23}$$

或另表示为：

$$[\dot{X}]=[A][X]+[B]\begin{bmatrix}v_l\\v_r\end{bmatrix}+[G]\begin{bmatrix}v_d\\\omega_d\end{bmatrix} \tag{11.24}$$

其中，

$$[A]=\begin{bmatrix}0&0&0\\v_c&0&0\\0&0&0\end{bmatrix} \tag{11.25}$$

$$[B]=\begin{bmatrix}\frac{1}{w}&\frac{-1}{w}\\0&0\\\frac{1}{2}&\frac{1}{2}\end{bmatrix} \tag{11.26}$$

$$[G]=\begin{bmatrix}0&1\\0&0\\-1&0\end{bmatrix} \tag{11.27}$$

其中$[A]$矩阵中$v_c=\frac{v_l+v_r}{2}$是机器人前进的速度。检查其是否为可控，可得可控矩阵：

$$\begin{aligned}[C_m]&=\begin{bmatrix}B & AB & A^2B\end{bmatrix}\\&=\begin{bmatrix}\frac{1}{w}&\frac{-1}{w}&0&0&0&0\\0&0&\frac{v_c}{w}&\frac{-v_c}{w}&0&0\\\frac{1}{2}&\frac{1}{2}&0&0&0&0\end{bmatrix}\end{aligned} \tag{11.28}$$

如果v_c不等于零，可控性矩阵的秩为3，表示此系统是可控的。然而此系统并不是可观测的，因为可观测矩阵为零矩阵。即：

$$[O_m]=\begin{bmatrix}C\\CA\\CA^2\end{bmatrix}=[0] \tag{11.29}$$

1. 路径追踪调节器的设计

由上述状态方程式，以及线/角速度与要求线/角速度的关系：

$$\omega_d=-\frac{v_l+v_r}{2} \tag{11.30}$$

$$v_d=\frac{v_l+v_r}{2} \tag{11.31}$$

可得：

$$\begin{bmatrix} v_l \\ v_r \end{bmatrix} = \begin{bmatrix} 1 & \frac{-w}{2} \\ 1 & \frac{w}{2} \end{bmatrix} \begin{bmatrix} v_d \\ \omega_d \end{bmatrix} = [F] \begin{bmatrix} v_d \\ \omega_d \end{bmatrix} \tag{11.32}$$

其中，

$$[F] = \begin{bmatrix} 1 & \frac{-w}{2} \\ 1 & \frac{w}{2} \end{bmatrix} \tag{11.33}$$

再组合状态方程式，可得如图 11-8 所示路径追踪控制器的状态空间模型。解这个状态方程式是一个调节器问题，3 个状态变量必须收敛到零。反馈矩阵 [K] 内的元素由控制器所需的性能决定。最佳控制中的线性二次调节器（Linear Quadratic Regulator，LQR）可以解决这个多变量问题。[A] 矩阵为一个时变矩阵，所以也是时变（Time Varying）问题。

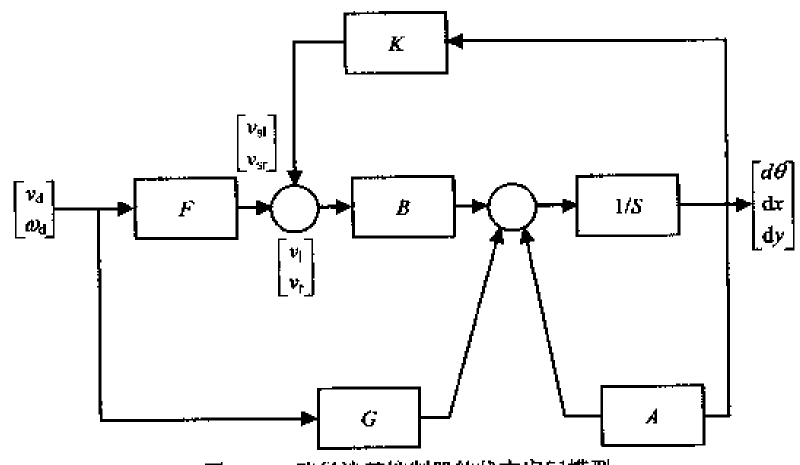


图 11-8　路径追踪控制器的状态空间模型

2. 时变最佳控制

下面我们介绍求反馈矩阵 [K] 的方法（参考 Franklin 所著的 Digital Control of Dynamic System)，对于一个离散时域的系统，其方程式表示为：

$$x(k+1) = \phi x(k) + \Gamma u(k) \tag{11.34}$$

要求决定 $u(k)$ 的值，使得成本函数（Cost Function）

$$J = \frac{1}{2}\sum_{k=0}^{N}\left[x^T(k)Q_1x(k) + u^T(k)Q_2u(k)\right] \tag{11.35}$$

的值达到最小。其中 Q_1、Q_2 为对称加权矩阵（Symmetric Weighting Matric)，需要由设计者决定且为非负定（Nonnegative Definite)，通常取其为对角矩阵（Diagonal Matrix)，对角值是正数或零。另一种表示法是要使下式最小：

$$J = \frac{1}{2}\sum_{k=0}^{N}\left[x^T(k)Q_1x(k) + u^T(k)Q_2u(k)\right] \tag{11.36}$$

其限制条件是

$$-x(k+1) + \phi x(k) + \Gamma u(k) = 0, k = 0,1,\cdots, N \tag{11.37}$$

求解此有限制最小值（Constrained Mimima）问题，可以应用拉格朗日乘数法，对于每

一个 k 我们定义一个拉格朗日乘数向量，然后可重写为：

$$J=\frac{1}{2}\sum_{k=0}^{N}\left[x^T(k)Q_1x(k)+u^T(k)Q_2u(k)+\phi^T(k+1)(-X(k+1)+\phi X(k)+\Gamma u(k)\right] \quad (11.38)$$

对于 $x(k)$、$u(k)$ 及 $\lambda(k)$ 找 $\hat{J}$ 的最小值

$$\frac{\partial\hat{J}}{\partial u(k)}=u^T(k)Q_2+\lambda^T(k+1)\Gamma=0 \quad (11.39)$$

$$\frac{\partial\hat{J}}{\partial\lambda(k+1)}=-x(k+1)+\phi x(k)+\Gamma u(k)=0 \quad (11.40)$$

$$\frac{\partial\hat{J}}{\partial x(k)}=x^T(k)Q_1-\lambda^T(k)+\lambda^T(k+1)\phi=0 \quad (11.41)$$

式（11.39）、式（11.40）、式（11.41）可分别写成式（11.42）、式（11.43）、式（11.44）：

$$u(k)=-Q_2^{-1}\Gamma^T\lambda(k+1) \quad (11.42)$$

$$x(k+1)=\phi x(k)+\Gamma u(k) \quad (11.43)$$

$$\lambda(k)=\phi^T\lambda(k+1)+Q_1x(k)=0 \quad (11.44)$$

式（11.44）也可以描述为前向差分方程：

$$\lambda(k+1)=\phi^{-T}\lambda(k)-\phi^{-T}x(k)Q_1 \quad (11.45)$$

假设初值或终值已知，式（11.42）、式（11.43）、式（11.45）为一组找最佳值 $x(k)$、$u(k)$、$\lambda(k)$ 的互偶差分方程组。又因为从式（11.40）中，知 $u(N)$ 对 $x(N)$ 没有作用，所以从式（11.36）知为使 $\hat{J}$ 达到最小，$u(N)$ 必须为零。再代入式（11.39），可得 $\lambda(N+1)=0$，再代入式（11.41）可以得到一边界条件：

$$\lambda(N)=Q_1x(N) \quad (11.46)$$

所以最佳控制的问题，可由两个差分方程式及给定的 u 及 λ 的边界条件 $\lambda(N)=Q_1x(N)$ 求出。而且 x 的初始条件必须先给定。对此两端点边界值问题是不易解得的，有一个方法称为 sweep 法，由 Bryson 及 Ho 提出，假设：

$$\lambda(k)=s(k)x(k) \quad (11.47)$$

代入式（11.39）得：

$$\begin{aligned}Q_2u(k)&=-\Gamma^Ts(k+1)x(k+1)\\&=-\Gamma^Ts(k+1)(\phi x(k)+\Gamma u(k))\end{aligned} \quad (11.48)$$

从上式解得：

$$\begin{aligned}u(k)&=-(Q_2+\Gamma^Ts(k+1)\Gamma)^{-1}\Gamma^Ts(k+1)\phi(k)\\&=-R^{-1}\Gamma^Ts(k+1)\phi x(k)\end{aligned} \quad (11.49)$$

在此定义 $R=Q_2+\Gamma^Ts(k+1)\Gamma$ 重写式（11.45）

$$\lambda(k)=\phi^T\lambda(k+1)+Q_1x(k) \quad (11.50)$$

将式（11.47）代入上式得：

$$\begin{aligned}s(k)x(k)&=\phi^Ts(k+1)x(k+1)+Q_1x(k)\\&=\phi^Ts(k+1)(\phi x(k)+\Gamma u(k))+Q_1x(k)\end{aligned} \quad (11.51)$$

将式（11.49）代入上式可得：

$$s(k)x(k)=\phi^T s(k+1)(\phi x(k)-\Gamma R^{-1}\Gamma^T s(k+1)\phi x(k))+Q_1 x(k) \tag{11.52}$$

移项合并可得：

$$(s(k)-\phi^T s(k+1)\phi+\phi^T s(k+1)\Gamma R^{-1}\Gamma^T s(k+1)\phi-Q_1)x(k)=0 \tag{11.53}$$

上式对所有的 $x(k)$ 均须成立，所以系数矩阵必须为零，可得：

$$\begin{aligned} s(k)&=\phi^T(s(k+1)-s(k+1)\Gamma R^{-1}\Gamma^T s(k+1))\phi+Q_1 \\ &=\phi^T M(k+1)\phi+Q_1 \end{aligned} \tag{11.54}$$

其中，

$$\begin{aligned} M(k+1)&=s(k+1)-s(k+1)\Gamma R^{-1}\Gamma^T s(k+1) \\ &=s(k+1)-s(k+1)\Gamma(Q_2+\Gamma^T s(k+1)\Gamma)^{-1}\Gamma^T s(k+1) \end{aligned} \tag{11.55}$$

上式称为离散 Riccati 方程式，再由式 $\lambda(k)=s(k)x(k)$ 及 $\lambda(N)=Q_1 x(N)$ 可得：

$$s(N)=Q_1 \tag{11.56}$$

至此，原来的问题已经转换为可由式（11.54）、（11.55）及单一边界条件式（11.56）来求解。这个迭代方程式必须由后往前解，因为边界条件给在终点可利用式（11.56）来解，即：

$$u(k)=-K(k)x(k) \tag{11.57}$$

其中，

$$K(k)=(Q_2+\Gamma^T s(k+1)\Gamma)^{-1}\Gamma^T s(k+1)\Phi \tag{11.58}$$

这就是所要求的时变最佳控制 $K(k)$。

上面我们介绍了由最小化成本函数来解的最佳控制增益矩阵 $K(k)$，它是一个时变控制增益矩阵，但是通常在整段的 $K(k)$ 解中会有一个保持不变的值 K_∞，这个值往往在控制系统的实时（Real Time）控制操作上要容易多了。实事上，对于无限时间的问题（称为调节器的情况）来说，这个不变的增益是最佳的。使用 MATLAB 的 dlqr()函数来获得稳态增益矩阵 $[K_\infty]$。由于系统矩阵 $[A]$ 为 v_c 的函数（见式 11.25），所以在每个采样时间（20ms）内，以当时的 v_c 调用 dlqr()函数来求得当时的 $[K_\infty]$，在此 20ms 内 v_c 视为固定值。

3. 计算机仿真

Q_1 和 Q_2 两矩阵的选择需要靠尝试错误法（Trail and Error），因为对 3 个误差变量同等重要，所以选择 Q_1 是一个主轴为 1 的对角矩阵，Q_2 的选择与控制量有关，也就与电机的速度有关。

为了得到较快的反应，在成本函数中加入 α 参数式（11.59），α 参数经尝试错误法后，选择 $\alpha=1.022$，有不错的响应。

$$J_\alpha=\sum_{k=0}^{\infty}\left[x^T(k)Q_1 x(k)+u^T(k)Q_2 u(t)\right]\alpha^{2k} \tag{11.59}$$

用以下几个步骤推导仿真用的程序：

（1）从状态方程式

$$\begin{bmatrix} \mathrm{d}\dot{\theta} \\ \mathrm{d}\dot{x} \\ \mathrm{d}\dot{y} \end{bmatrix}=\begin{bmatrix} 0 & 0 & 0 \\ v_c & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} \mathrm{d}\theta \\ \mathrm{d}x \\ \mathrm{d}y \end{bmatrix}+\begin{bmatrix} \frac{1}{w} & \frac{-1}{w} \\ 0 & 0 \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}\begin{bmatrix} v_l \\ v_r \end{bmatrix}+\begin{bmatrix} 0 & 1 \\ 0 & 0 \\ -1 & 0 \end{bmatrix}\begin{bmatrix} v_d \\ \omega_d \end{bmatrix} \tag{11.60}$$

$$[A]=\begin{bmatrix}0 & 0 & 0\\ v_c & 0 & 0\\ 0 & 0 & 0\end{bmatrix} \tag{11.61}$$

$$[B]=\begin{bmatrix}\frac{1}{w} & \frac{-1}{w}\\ 0 & 0\\ \frac{1}{2} & \frac{1}{2}\end{bmatrix} \tag{11.62}$$

$$[G]=\begin{bmatrix}0 & 1\\ 0 & 0\\ -1 & 0\end{bmatrix} \tag{11.63}$$

和选择

$$Q_1=\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix} \tag{11.64}$$

$$Q_2=\begin{bmatrix}\frac{1}{2} & 0\\ 0 & \frac{1}{2}\end{bmatrix} \tag{11.65}$$

（2）MATLAB 中的 c2d()函数，可将状态方程式从连续时域变换为离散时域

$$\begin{aligned}[Ad,Bd]&=c2d(A,B,T)\\ [Cd,Gd]&=c2d(A,G,T)\end{aligned} \tag{11.66}$$

T 为采样周期，即离散状态方程式为：

$$\begin{bmatrix}\Delta\theta(k+1)\\ \Delta x(k+1)\\ \Delta y(k+1)\end{bmatrix}=[Ad]\begin{bmatrix}\theta(k)\\ \Delta x(k)\\ \Delta y(k)\end{bmatrix}+[Bd]\begin{bmatrix}v_1(k)\\ v_r(k)\end{bmatrix}+[Gd]\begin{bmatrix}v_d\\ \omega_d\end{bmatrix} \tag{11.67}$$

（3）利用 MATLAB 的 dlqr( )函数来计算$[K]$

$$[K]=\text{dlqr}(Ad*1.022,Bd*1.022,Q_1,Q_2) \tag{11.68}$$

（4）计算反馈修正速度

$$\begin{bmatrix}v_{sl}(k)\\ v_{sr}(k)\end{bmatrix}=-[K(k)]\begin{bmatrix}\Delta\theta(k)\\ \Delta x(k)\\ \Delta y(k)\end{bmatrix}=-\begin{bmatrix}k_1\theta(k) & k_1x(k) & k_1y(k)\\ k_2\theta(k) & k_2x(k) & k_2y(k)\end{bmatrix}\begin{bmatrix}\Delta\theta(k)\\ \Delta x(k)\\ \Delta y(k)\end{bmatrix} \tag{11.69}$$

（5）计算需求速度

$$\begin{bmatrix} v_{dl}(k) \\ v_{dr}(k) \end{bmatrix} = [F]\begin{bmatrix} v_d \\ \omega_d \end{bmatrix} + \begin{bmatrix} v_{sl}(k) \\ v_{sr}(k) \end{bmatrix} = \begin{bmatrix} 1 & \dfrac{-w}{2} \\ 1 & \dfrac{w}{2} \end{bmatrix}\begin{bmatrix} v_d \\ \omega_d \end{bmatrix} + \begin{bmatrix} v_{sl}(k) \\ v_{sr}(k) \end{bmatrix} \tag{11.70}$$

其中，

- $v_{dl}(k)$为 k 时左轮的要求速度
- $v_{dr}(k)$为 k 时右轮的要求速度

（6）仿真时，假设电机有理想的响应速度即

$$v_l(k+1) = v_{dl}(k) \tag{11.71}$$

$$v_r(k+1) = v_{dr}(k) \tag{11.72}$$

其中，

- $v_l(k+1)$为 k+1 时左轮的要求速度
- $v_r(k+1)$为 k+1 时右轮的要求速度

（7）计算

$$v_c(k+1) = \frac{1}{2}(v_l(k+1) + v_r(k+1)) \tag{11.73}$$

$$\omega_c(k+1) = \frac{1}{w}(v_r(k+1) - v_l(k+1)) \tag{11.74}$$

至步骤（2）计算$\Delta\theta(k+1)$，$\Delta x(k+1)$，$\Delta y(k+1)$。

4. 仿真结果

应用以上所述，经过计算机中 MATLAB 仿真来观察不同起始误差条件下路径追踪控制器的效果。控制器的采样时间为 20ms，仿真时假设加速度无穷大，即左右轮速度等于左右轮速度命令。结果发现经过一段时间即由 LQR 路径追踪控制器的速度即角速度的补偿而消除 3 个追踪误差量。

在 MATLAB 中执行以下程序 ch11_1.m

```
% ch11_1.m
% LQR 路径追踪控制系统
deltaX(1)=2 ; deltaY(1)=2 ; deltaS(1)=0.0349;
Vd=10 ; Wd=0.052;
k=1;vv(k)=Vd;
A=[0 0 0;vv(k) 0 0;0 0 0];
B=[1/67.5 -1/67.5 ; 0 0 ; 0.5 0.5];
G=[0 1;0 0;-1 0];
Q1=[1 0 0;0 1 0;0 0 1];
Q2=[0.5 0;0 0.5];
Ts=0.02 ; F=[1 -33.75;1 33.75];
x(1)=100;y(1)=100;
x1(1)=x(1)+deltaX(1);y1(1)=y(1)+deltaY(1);
Vdl(1)=Vd-0.5*67.5*Wd;
```

```
Vdr(1)=Vd+0.5*67.5*Wd;
for k=1:400
 err=[deltaS(k)
      deltaX(k)
      deltaY(k)];
 if (k==1)
    vv(k)=Vd;
 end
 [ad,bd]=c2d(A,B,Ts);
 [cd,dd]=c2d(A,G,Ts);

 K=dlqr(ad*1.022,bd*1.022,Q1,Q2);

 k1S(k)=K(1,1);
 k2S(k)=K(2,1);
 k1X(k)=K(1,2);
 k2X(k)=K(2,2);
 k1Y(k)=K(1,3);
 k2Y(k)=K(2,3);

 Vsl=-K(1,:)*err;
 Vsr=-K(2,:)*err;
 Vdl(k+1)=[1 -33.75]*[Vd;Wd]+Vsl;
 Vdr(k+1)=[1 33.75]*[Vd;Wd]+Vsr;
 vv(k+1)=0.5*(Vdl(k+1)+Vdr(k+1));
 if (Vdl(k+1)>45.0)
    Vdl(k+1)=45.0;
 elseif (Vdl(k+1)<-45.0)
    Vdl(k+1)=-45.0;
 end
 if (Vdr(k+1)>45.0)
    Vdr(k+1)=45.0;
 elseif (Vdr(k+1)<-45.0)
    Vdr(k+1)=-45.0;
 end
 deltaS(k+1)=ad(1,:)*err+bd(1,:)*[Vdl(k+1);Vdr(k+1)]+dd(1,:)*[Vd;Wd];
 deltaX(k+1)=ad(2,:)*err+bd(2,:)*[Vdl(k+1);Vdr(k+1)]+dd(2,:)*[Vd;Wd];
 deltaY(k+1)=ad(3,:)*err+bd(3,:)*[Vdl(k+1);Vdr(k+1)]+dd(3,:)*[Vd;Wd];

end
```

```
n=k+1;
for i=1:n
k(i)=i*Ts;
end
subplot(221),plot(k,deltaX)
xlabel('秒');ylabel('侧向误差')
grid
axis([0 10 -0.5 3])
subplot(222),plot(k,deltaY)
xlabel('秒');ylabel('前向误差')
grid
axis([0 10 -0.5 3])
subplot(223),plot(k,deltaS*57.29)
xlabel('秒');ylabel('指向误差')
grid
axis([0 10 -12 3])
subplot(224),plot(k,Vdl,'-',k,Vdr,'.')
xlabel('秒')
ylabel('左右轮速度（cm/s）')
gtext('右轮')
grid
```

如图 11-9 所示图形为初始误差 $dx=2\text{cm}$，$dy=2\text{cm}$，$d\theta=2^\circ$，$\omega_d=0$及$v_d=10\text{cm/s}$的仿真结果，误差一开始作为阶跃输入，然后观察误差衰减情形及左右轮速度变化图。图 11-10 初始误差同图 11-9，但$\omega_d=0.052\text{rad/s}$。

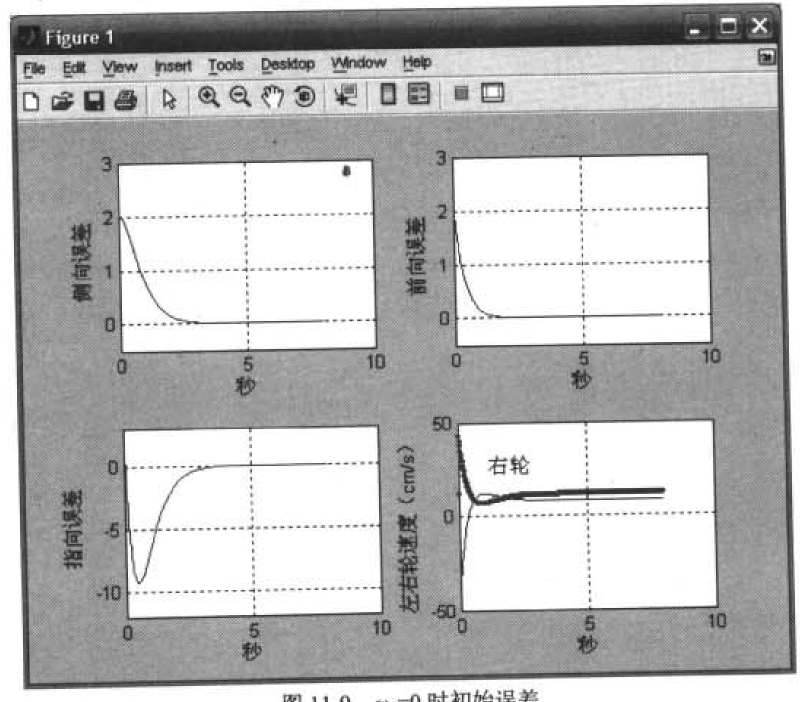


图 11-9　ω_d=0 时初始误差

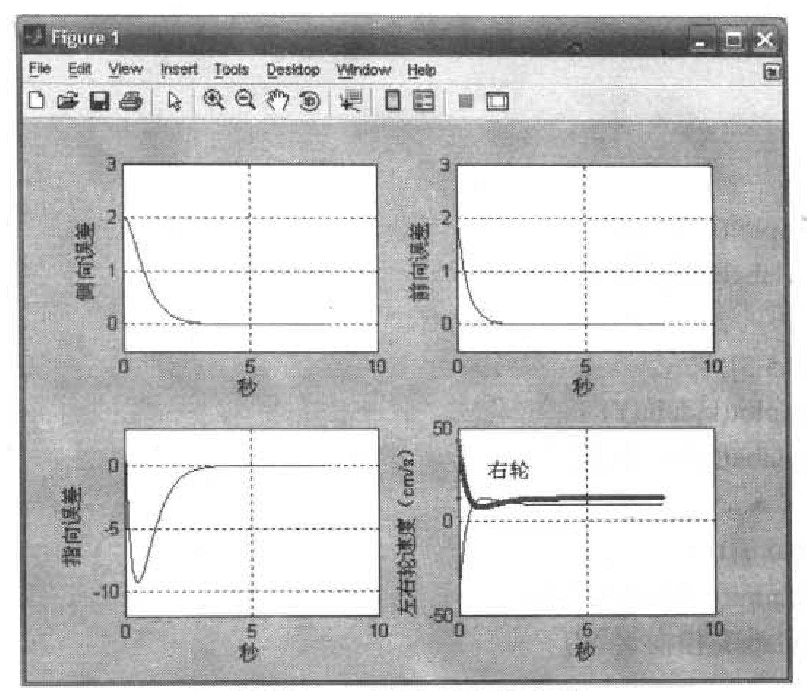


图 11-10 ω_d=0.052rad/s 时初始误差

参 考 文 献

1. 皇甫堪. 现代数字信号处理. 长沙: 国防科技大学出版社, 2003
2. 刘福声等. 统计信号处理. 长沙: 国防科技大学出版社, 1999
3. 王沫然. Simulink4 建模与动态仿真. 北京: 电子工业出版社, 2001
4. 陈桂明. 应用 MATLAB 建模与仿真. 北京: 科学技术出版社, 2001
5. 苏金明. MATLAB 7.0 实用指南. 北京: 电子工业出版社, 2005
6. 邹 鲲. MATLAB 6.x 信号处理. 北京: 清华大学出版社, 2002
7. 何衍庆. 控制系统分析、设计和应用: MATLAB 语言的应用. 北京: 化学工业出版社, 2003
8. 王 宏. MATLAB 6.5 及其在信号处理中的应用. 北京: 清华大学出版社, 2004
9. 张 森. MATLAB 仿真技术与实例应用教程. 北京: 机械工业出版社, 2004
10. 张葛祥. MATLAB 仿真技术与应用. 北京: 清华大学出版社, 2003
11. 钟 麟. MATLAB 仿真技术与应用教程. 北京: 国防工业出版社, 2004
12. 苏金明. MATLAB 工具箱应用. 北京: 电子工业出版社, 2004
13. 陈怀琛. MATLAB 及在电子信息课程中的应用. 北京: 电子工业出版社, 2003
14. 魏 巍. MATLAB 控制工程工具箱技术手册. 北京: 国防工业出版社, 2004
15. 欧阳黎明. MATLAB 控制系统设计. 北京: 国防工业出版社, 2001
16. 孙 屹. MATLAB 通信仿真开发手册. 北京: 国防工业出版社, 2005
17. 刘 敏. MATLAB 通信仿真与应用. 北京: 国防工业出版社, 2001
18. 魏 巍. MATLAB 信息工程工具箱技术手册. 北京: 国防工业出版社, 2004
19. 孙 亮. MATLAB 语言与控制系统仿真. 北京: 北京工业大学出版社, 2001
20. 魏克新. MATLAB 语言与自动控制系统设计. 北京: 机械工业出版社, 1997
21. 薛年喜. MATLAB 在数字信号处理中的应用. 北京: 清华大学出版社, 2003
22. 黄忠霖. 控制系统 MATLAB 计算及仿真. 北京: 国防工业出版社, 2001
23. William J Palm. Introduction to MATLAB 7 for Engineers, 2004
24. Jim Ledin. Simulation Engineering, 2001
25. William S Levine. Control System Fundamentals, 1999
26. Jim Ledin. Simulation Engineering. CMP Books, 2001
27. Mathworks. MATLAB 6.5, 2002
28. Mathworks. MATLAB 7.0, 2004